man pages section 7: Device and Network Interfaces
Contents

Preface 11

Introduction 17
Intro(7) 18

Device and Network Interfaces 21
adp(7D) 22
afb(7d) 23
arp(7P) 24
asy(7D) 27
ata(7D) 30
audio(7I) 32
audioamd(7D) 43
audiocs(7D) 45
audio_support(7I) 49
audiots(7D) 51
authmd5h(7M) 55
authsha1(7M) 56
bd(7M) 57
bpp(7D) 59
bufmod(7M) 64
bwtwo(7D) 68
cadp160(7D) 69
cadp(7D) 70
cdio(7I) 73
ce(7D)  81
ceight(7D)  85
cgfour(7D)  86
cgfourteen(7D)  87
cgsix(7D)  88
cgtthree(7D)  89
cgtwo(7D)  90
cmdk(7D)  91
cnft(7D)  92
connld(7M)  96
console(7D)  98
cpqnrcr(7D)  99
cpr(7)  100
cvc(7D)  102
cvcredir(7D)  103
dad(7D)  104
dbri(7D)  107
devinfo(7D)  113
dkio(7I)  114
dlpi(7P)  123
dman(7D)  124
dmfe(7D)  127
dnet(7D)  129
dr(7D)  131
ecpp(7D)  133
ehx(7D)  139
ehx(7D)  141
encr3des(7M)  143
enrcdes(7M)  144
eri(7D)  145
esp(7D)  149
fas(7D)  155
fbio(7I)  163
fcip(7D)  165
fcp(7D)  168
fcntl(7D)  169
fd(7D)  170
fdio(7I) 176
ffb(7D) 180
flashpt(7D) 181
fp(7d) 182
g(7D) 183
gld(7D) 187
glm(7D) 195
gpio_87317(7D) 200
hci1394(7D) 201
hdio(7I) 202
hid(7D) 204
hme(7D) 206
hpfc(7D) 211
hsfs(7FS) 213
hubd(7D) 216
i2o_bs(7D) 218
i2o_scsi(7D) 220
icmp6(7P) 221
icmp(7P) 223
idn(7d) 225
ifb(7d) 228
ifp(7D) 229
if_tcp(7P) 233
inet6(7P) 240
inet(7P) 243
ip6(7P) 246
ip(7P) 252
iprb(7D) 257
ipsec(7P) 260
ipsecah(7P) 263
ipsecesp(7P) 264
isdnio(7I) 266
isp(7D) 280
kb(7M) 286
kdmouse(7D) 295
kstat(7D) 296
ksyms(7D) 297
<table>
<thead>
<tr>
<th>man page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldterm(7M)</td>
<td>299</td>
</tr>
<tr>
<td>le(7D)</td>
<td>302</td>
</tr>
<tr>
<td>lic1(7D)</td>
<td>306</td>
</tr>
<tr>
<td>lic2(7D)</td>
<td>309</td>
</tr>
<tr>
<td>lockstat(7D)</td>
<td>316</td>
</tr>
<tr>
<td>lofi(7D)</td>
<td>317</td>
</tr>
<tr>
<td>lofs(7FS)</td>
<td>318</td>
</tr>
<tr>
<td>log(7D)</td>
<td>320</td>
</tr>
<tr>
<td>logi(7D)</td>
<td>324</td>
</tr>
<tr>
<td>lp(7D)</td>
<td>325</td>
</tr>
<tr>
<td>litem(7D)</td>
<td>327</td>
</tr>
<tr>
<td>m64(7D)</td>
<td>328</td>
</tr>
<tr>
<td>mem(7D)</td>
<td>329</td>
</tr>
<tr>
<td>mhd(7i)</td>
<td>330</td>
</tr>
<tr>
<td>mixer(7I)</td>
<td>335</td>
</tr>
<tr>
<td>msglog(7D)</td>
<td>345</td>
</tr>
<tr>
<td>msm(7D)</td>
<td>346</td>
</tr>
<tr>
<td>mt(7D)</td>
<td>347</td>
</tr>
<tr>
<td>mtio(7I)</td>
<td>348</td>
</tr>
<tr>
<td>ncrs(7D)</td>
<td>360</td>
</tr>
<tr>
<td>null(7D)</td>
<td>367</td>
</tr>
<tr>
<td>ocf_escr1(7D)</td>
<td>368</td>
</tr>
<tr>
<td>ocf_ibutton(7D)</td>
<td>369</td>
</tr>
<tr>
<td>ocf_iscr1(7D)</td>
<td>370</td>
</tr>
<tr>
<td>ohci(7D)</td>
<td>371</td>
</tr>
<tr>
<td>openprom(7D)</td>
<td>372</td>
</tr>
<tr>
<td>pcata(7D)</td>
<td>377</td>
</tr>
<tr>
<td>pcclx(7D)</td>
<td>378</td>
</tr>
<tr>
<td>pcfis(7FS)</td>
<td>379</td>
</tr>
<tr>
<td>pccic(7D)</td>
<td>384</td>
</tr>
<tr>
<td>pckt(7M)</td>
<td>385</td>
</tr>
<tr>
<td>pcmem(7D)</td>
<td>386</td>
</tr>
<tr>
<td>pcn(7D)</td>
<td>387</td>
</tr>
<tr>
<td>pcram(7D)</td>
<td>389</td>
</tr>
<tr>
<td>pccssi(7D)</td>
<td>390</td>
</tr>
<tr>
<td>pcser(7D)</td>
<td>391</td>
</tr>
<tr>
<td>pf_key(7P)</td>
<td>392</td>
</tr>
</tbody>
</table>
sockio(7I) 497
spwr(7D) 498
ssd(7D) 499
st(7D) 504
stc(7D) 516
stp4020(7D) 528
streamio(7I) 529
sxp(7D) 545
symhisl(7D) 548
sysmsg(7D) 552
t1394dcam(7D) 553
tcp(7P) 554
tcx(7D) 559
termio(7I) 561
termiox(7I) 582
ticlts(7D) 588
timod(7M) 590
tirdwr(7M) 592	tmpfs(7FS) 594
tpf(7D) 596
ttcompat(7M) 597
tty(7D) 604
tun(7M) 605
uta(7D) 609
udfs(7FS) 611
udp(7P) 612
uhci(7D) 615
usba(7D) 616
usb_ac(7D) 618
usb_as(7D) 622
usbkbm(7M) 624
usb_mid(7D) 626
usbms(7M) 627
usbprn(7D) 629
uscsi(7I) 635
usoc(7D) 639
visual_io(7I) 642
volfs(7FS) 649
vuidmice(7M) 651
wrsm(7D) 654
wrsmd(7D) 656
wscons(7D) 657
xd(7D) 666
xmemfs(7FS) 668
xt(7D) 670
xy(7D) 672
zero(7D) 674
zs(7D) 675
zsh(7D) 678

Index 683
Preface

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

Overview

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

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 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 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 can include in a device driver.

Section 9F describes the kernel functions available for use by device drivers.

Section 9S describes the kernel functions used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

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

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

The following special characters are used in this section:

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

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

| Separator. Only one of the arguments separated by this character can be specified at a time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL
This section occurs only in subsection 3R to indicate the protocol description file.

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

IOCTL
This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mtio(7I).

OPTIONS
This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS
This section lists the command operands and describes how they affect the actions of the command.

OUTPUT
This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS
On failure, most functions place an error code in the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than
one condition can cause the same error, each condition is described in a separate paragraph under the error code.

**USAGE**

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- **Commands**
- **Modifiers**
- **Variables**
- **Expressions**
- **Input Grammar**

**EXAMPLES**

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

**ENVIRONMENT VARIABLES**

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

**EXIT STATUS**

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

**FILES**

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

**ATTRIBUTES**

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See attributes(5) for more information.

**SEE ALSO**

This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS  This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS  This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES  This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS  This section describes known bugs and, wherever possible, suggests workarounds.
Introduction
This section describes various device and network interfaces available on the system. The types of interfaces described include character and block devices, STREAMS modules, network protocols, file systems, and ioctl requests for driver subsystems and classes.

This section contains the following major collections:

(7D)

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

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

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

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

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

Some driver configuration files may contain user configurable properties. Changes in a driver's configuration file will not take effect until the system is rebooted or the driver has been removed and re-added (see rem_drv(1M) and add_drv(1M)).
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 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. 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 inet(7P), is the primary protocol family supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the AF_INET address family when binding a socket; see socket(3SOCKET) for details.

Major protocols in the Internet family include:

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

SEE ALSO

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

Solaris Transition Guide

System Administration Guide, Volume 3

STREAMS Programming Guide
Device and Network Interfaces
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 is (see NOTES):

<table>
<thead>
<tr>
<th>AIC-7560</th>
<th>AIC-7870</th>
<th>AIC-7881</th>
</tr>
</thead>
<tbody>
<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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)

Hardware Compatibility List for Solaris 2.6 (Intel Platform Edition)

Solaris 8 (Intel Platform Edition) Installation Guide

NOTES

Throughout the release, support of additional devices may be added. See the Hardware Compatibility List for Solaris 2.6 (Intel Platform Edition) in the Solaris 8 (Intel Platform Edition) Installation Guide for additional information.

The adp driver supports Logical Unit Number (“LUN”) values of 0 through 15, this is beyond the standard SCSI-2 requirements which call for support of LUNs 0 through 7.
The `afb` driver is the device driver for the Sun Elite3D graphics accelerators. The `afbdaemon` process loads the `afb` microcode at system startup time and during the resume sequence of a suspend-resume cycle.

<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>afb</code></td>
<td>- Elite3D graphics accelerator driver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/dev/fbs/afb</code></td>
<td>Device special file</td>
</tr>
<tr>
<td><code>/usr/lib/afb.ucode</code></td>
<td><code>afb</code> microcode</td>
</tr>
<tr>
<td><code>/usr/sbin/afbdaemon</code></td>
<td><code>afb</code> microcode loader</td>
</tr>
</tbody>
</table>

**SEE ALSO** `afbconfig(1M)`
arp(7P)

NAME  arp, ARP – Address Resolution Protocol

SYNOPSIS  

```
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
d = open("/dev/arp", oflag);
```

DESCRIPTION  ARP is a protocol used to map dynamically between Internet Protocol (IP) and 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. The only network layer supported in this implementation is the Internet Protocol, although ARP is not specific to that 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, ARP caches the new mapping and transmits any pending message. ARP will queue at most four packets while waiting for a response to a mapping request; it keeps only the four most recently transmitted packets.

APPLICATION PROGRAMMING INTERFACE  

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

To facilitate communications with systems that do not use ARP, ioctl() requests are provided to enter and delete entries in the IP-to-Ethernet tables.

```
#include <sys/socket.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.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 s, or to a descriptor for the ARP device, but only by the privileged user.

The arpreq structure contains:

```
/*
 * ARP ioctl request
*/
struct arpreq {
```

24  man pages section 7: Device and Network Interfaces • Last Revised 23 Aug 1994
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; trailer encapsulations are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either host or both may request trailers. The `ATF_USETRAILERS` flag records 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.%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 '%d.%d.%d.%d'
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 ARPentry.
asy(7D)

NAME
asy – asynchronous serial port driver

SYNOPSIS
#include <fcntl.h>
#include <sys/termios.h>

open("/dev/tty\nn", mode);
open("/dev/ttyd\nn", mode);
open("/dev/cua\nn", 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 and 16550 hardware, in addition to basic asynchronous communication support. The asy module supports those termio(7I) device control functions specified by flags in the c_cflag word of the termios structure, and by the IGNBRK, IGNPAR, FPARMRK, 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 module supports up to four serial ports, including the standard ports. The tty\nn devices have minor device numbers in the range 00-03, and may be assigned names of the form /dev/ttydn, where \nn denotes the line to be accessed. These device names are typically used to provide a logical access point for a dial-in line that is used with a modem.

To allow a single tty line to be connected to a modem and used for incoming and outgoing calls, a special feature is available that is controlled by the minor device number. By accessing character-special devices with names of the form /dev/cuan, 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/cuan line is opened, the corresponding tty, or ttyd line cannot be opened until the /dev/cuan line is closed. A blocking open will wait until the /dev/cuan line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again. A non-blocking open will return an error. If the /dev/ttydn line has been opened successfully (usually only when carrier is recognized on the modem), the corresponding /dev/cuan line cannot be opened. This allows a modem to be attached to a device, (for example, /dev/ttyd0, which is renamed from /dev/tty00 and used for dial-in (by enabling the line for login in /etc/inittab) or dial-out (by tip(1) or uucp(1C)) as /dev/cua0 when no one is logged in on the line.

APPLICATION

PROGRAMMING

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

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

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

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

A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. The Alternate Break sequence can be used as a remedy against this. Due to a risk of incorrect sequence interpretation, binary protocols such as PPP, SLIP, and others should not be run over the serial console port when Alternate Break sequence is in effect. By default, the Alternate Break sequence is a three character sequence: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For more information on breaking (entering the debugger or monitor), see `kbd(1)` and `kb(7M)`.

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

- **ENXIO** The unit being opened does not exist.
- **EBUSY** The dial-out device is being opened while the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
- **EBUSY** The unit has been marked as exclusive-use by another process with a `TIOCEXCL ioctl()` call.
- **EINTR** The open was interrupted by the delivery of a signal.

### FILES

- `/dev/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>IA</td>
</tr>
</tbody>
</table>
SEE ALSO  tip(1), kbd(1), uucp(1C), ioctl(2), open(2), termios(3C), attributes(5), ldterm(7M), ttcompat(7M), kb(7M) termio(7I)

DIAGNOSTICS  

asy: silo overflow.
    The hardware overrun occurred before the input character could be serviced.

asy: ring buffer overflow.
    The driver’s character input ring buffer overflowed before it could be serviced.
ata – AT attachment disk driver

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.

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
- max_transfer

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 IA 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 Sample ata 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" ;
```
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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), cmdk(7D)
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 \texttt{AUDIO\_GETDEV\_ioctl(2)} to determine the current audio hardware associated with \texttt{/dev/audio}.

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

Audio data may be represented in several different formats. An audio device’s current audio data format can be determined by using the \texttt{AUDIO\_GETINFO\_ioctl(2)} described below.

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

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

### Encodings
An encoding parameter specifies the audio data representation. µ-Law encoding corresponds to \textit{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 \textit{CCITT G.711} and is the standard encoding for telephony elsewhere in the world. A-Law and µ-Law audio data are sampled at a rate of 8000 samples per second with 12-bit precision, with the data compressed to 8-bit samples. The resulting audio data quality is equivalent to that of standard analog telephone service.

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

### Precision
Precision indicates the number of bits used to store each audio sample. For instance, µu-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.
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.

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.

Alternatively, opening /dev/audio may open a mixing audio driver that provides a super set of this audio interface. The audio mixer removes the exclusive resource restriction, allowing multiple processes to play and record audio at the same time. See the mixer(7I) and audio_support(7I) manual pages for more information.

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

The audio device is treated as an exclusive resource — only one process can open the device at a time. However, if the AUDIO_DUBLEX bit is set in hw_features of the audio_info_t structure, two processes may simultaneously access the device. This allows one process to open the device as read-only and a second process to open it as write-only. See below for details.

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

- if either the O_NDELAY or O_NONBLOCK flags are set in the open() flag 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 μ-Law data. If the device is already open and a different audio format has been set, this will not be possible on some devices. 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.
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.

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

Although `write()` returns when the data is successfully queued, the actual completion of audio output may take considerably longer. The `AUDIO_DRAIN ioctl` may be issued to allow an application to block until all of the queued output data has been played. Alternatively, a process may request asynchronous notification of output completion by writing a zero-length buffer (end-of-file record) to the output stream. When such a buffer has been processed, the `play.eof` flag in the audio information structure (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 \texttt{AUDIO\_SETINFO} ioctl to set (or clear) the \texttt{play.pause} flag in the audio information structure. Queued output data may be discarded by using the \texttt{I\_FLUSH STREAMS} ioctl. (See \texttt{streamio(7I)}).

Output data will be played from the STREAMS buffers at a rate of at least 8000 bytes per second for \texttt{µ-Law} or \texttt{A-Law} data (faster for 16-bit linear data or higher sampling rates). If the output queue becomes empty, the \texttt{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 must be closed before any future writes will succeed.

Asynchronous I/O

The \texttt{I\_SETSIG STREAMS} ioctl enables asynchronous notification, through the \texttt{SIGPOLL} signal, of input and output ready conditions. The \texttt{O\_NONBLOCK} flag may be set using the \texttt{F\_SETFL} \texttt{fcntl(2)} to enable non-blocking \texttt{read()} and \texttt{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 \texttt{/dev/audioctl} pseudo-device is provided for this purpose. Any number of processes may open \texttt{/dev/audioctl} simultaneously. However, \texttt{read()} and \texttt{write()} system calls are ignored by \texttt{/dev/audioctl}. The \texttt{AUDIO\_GETINFO} and \texttt{AUDIO\_SETINFO} ioctl commands may be issued to \texttt{/dev/audioctl} to determine the status or alter the behavior of \texttt{/dev/audio}. Note: In general, the audio control device name is constructed by appending the letters "ctl" to the path name of the audio device.

Audio Status Change Notification

Applications that open the audio control pseudo-device may request asynchronous notification of changes in the state of the audio device by setting the \texttt{S\_MSG} flag in an \texttt{I\_SETSIG STREAMS} ioctl. Such processes receive a \texttt{SIGPOLL} signal when any of the following events occur:

- An \texttt{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 \texttt{open()} or \texttt{close()} of \texttt{/dev/audio} has altered the device state.
- An external event (such as speakerbox volume control) has altered the device state.
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 */
struct audio_prinfo {
    /* The following values describe the audio data encoding */
    uint_t sample_rate; /* samples per second */
    uint_t channels; /* number of interleaved channels */
    uint_t precision; /* number of bits per sample */
    uint_t encoding; /* data encoding method */

    /* The following values control audio device configuration */
    uint_t gain; /* volume level */
    uint_t port; /* selected I/O port */
    uint_t buffer_size; /* I/O buffer size */

    /* The following values describe the current device state */
    uint_t samples; /* number of samples converted */
    uint_t eof; /* End Of File counter (play only) */
    uchar_t pause; /* non-zero if paused, zero to resume */
    uchar_t error; /* non-zero if overflow/underflow */
    uchar_t waiting; /* non-zero if a process wants access */
    uchar_t balance; /* stereo channel balance */

    /* The following values are read-only device state information */
    uchar_t open; /* non-zero if open access granted */
    uchar_t active; /* non-zero if I/O active */
    uint_t avail_ports; /* available I/O ports */
    uint_t mod_ports; /* modifyable 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 */
    uint_t hw_features; /* supported H/W features */
    uint_t sw_features; /* supported S/W features */
    uint_t sw_features_enabled; /* supported S/W features enabled */
} audio_info_t;

/* Audio encoding types */
#define AUDIO_ENCODING_ULAW (1) /* u-Law encoding */
#define AUDIO_ENCODING_ALAW (2) /* A-Law encoding */
#define AUDIO_ENCODING_LINEAR (3) /* Signed Linear PCM encoding */

/* These ranges apply to record, play, and monitor gain values */
#define AUDIO_MIN_GAIN (0) /* minimum gain value */
#define AUDIO_MAX_GAIN (255) /* maximum gain value */

/* These values apply to the balance field to adjust channel gain values */
#define AUDIO_LEFT_BALANCE (0) /* left channel only */
```
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), `AUDIO_LINE_OUT` (line-out port), `AUDIO_AUX1_OUT` (auxiliary1 out), or `AUDIO_AUX2_OUT` (auxiliary2 out). For some devices, it may be set to a combination of

```c
typedef struct audio_device {
    char name[MAX_AUDIO_DEV_LEN];
    char version[MAX_AUDIO_DEV_LEN];
    char config[MAX_AUDIO_DEV_LEN];
} audio_device_t;
```
these ports. The `playavail_ports` field returns the set of output ports that are currently accessible. The `playmod_ports` field returns the set of output ports that may be turned on and off. If a port is missing from `playmod_ports` then that port is assumed to always be on.

The input ports can be either `AUDIO_MICROPHONE` (microphone jack), `AUDIO_LINE_IN` (line-out port), `AUDIO_CD` (internal CD-ROM), `AUDIO_AUX1_IN` (auxiliary1 in), `AUDIO_AUX2_IN` (auxiliary2 in), or `AUDIO_CODEC_LOOPB_IN` (internal loopback). The `recordavail_ports` field returns the set of input ports that are currently accessible. The `recordmod_ports` field returns the set of input ports that may be turned on and off. If a port is missing from `recordmod_ports`, it is assumed to always be on. Input ports are considered to be mutually exclusive.

The `playbalance` and `recordbalance` 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 `playpause` and `recordpause` flags may be used to pause and resume the transfer of data between the audio device and the STREAMS buffers. The `playerror` and `recorderror` flags indicate that data underflow or overflow has occurred. The `playactive` and `recordactive` flags indicate that data transfer is currently active in the corresponding direction.

The `playopen` and `recordopen` flags indicate that the device is currently open with the corresponding access permission. The `playwaiting` and `recordwaiting` 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 `playsamples` and `recordsamples` 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 `playeof` 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 `recordbuffersize` 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. Some audio
hardware may constrain the input and output data formats to be identical. If this is the
case, the data format may not be changed if multiple processes have opened the audio
device. As a result, a process should check that the ioctl() does not fail when it
attempts to set the data format.

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.

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.

The audio device additionally supports the following ioctl commands:

audio(7I)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO_DRAIN</td>
<td>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.</td>
</tr>
<tr>
<td>AUDIO_GETDEV</td>
<td>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</td>
</tr>
</tbody>
</table>
string that will identify the current /dev/audio hardware device, the value in version will be a string indicating the current version of the hardware, and config will be a device-specific string identifying the properties of the audio stream associated with that file descriptor. Refer to the audio device-specific manual pages to determine the actual strings returned by the device driver.

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

**AUDIO_SETINFO**  The argument is a pointer to an audio_info structure. This command may be issued for either the /dev/audio or the /dev/audiocpl device with some restrictions. This command configures the audio device according to the supplied structure and overwrites the existing structure with the new state of the device. Note: The play.samples, record.samples, play.error, record.error, and play.eof fields are modified to reflect the state of the device when the AUDIO_SETINFO 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/audiocpl, 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, play.port, record.port and monitor_gain. 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_t structure may be initialized through the use of the AUDIO_INITINFO macro. This macro sets all fields in the structure to values that are ignored by the AUDIO_SETINFO command. For instance, the following code switches the output port from the built-in speaker to the headphone jack without modifying any other audio parameters:

```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.

**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`
- `/usr/share/audio` audio files

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>All</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcsu, SUNWaudd, SUNWauddx, SUNWaudh</td>
</tr>
<tr>
<td>Stability Level</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`close(2), fcntl(2), ioctl(2), open(2), poll(2), read(2), write(2), audiocs(7D), dbri(7D), sbpro(7D), usb_ac(7D), audio_support(7I), mixer(7I), streamio(7I)`

**BUGS**
Due to a feature of the STREAMS implementation, programs that are terminated or exit without closing the audio device may hang for a short period while audio output drains. In general, programs that produce audio output should catch the SIGINT signal and flush the output stream before exiting.
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 audio drivers should use the mixer(7I) audio device to gain access to these new features.
audioamd(7D)

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.

Supported Audio Data Formats

<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</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 record.avail_ports and play.avail_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.
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**

`ioct1(2), attributes(5), audio(7I), streamio(7I)`

AMD data sheet for the AM79C30A Digital Subscriber Controller, Publication number 09893.
The audiocs devices uses the Crystal Semiconductor 4231 Codec to implement the audio device interface. This interface is described in the mixer(7I) and audio(7I) man pages.

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 a letter, defined below and the config field will contain the string onboard1.

<table>
<thead>
<tr>
<th>Platform Type</th>
<th>Version</th>
<th>Line</th>
<th>Headphone</th>
<th>Internal</th>
<th>Line</th>
<th>Mic</th>
<th>CD-ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS-4/5 a</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Ultra-1/2 b</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Reserved c</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>PowerPC d</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Reserved e</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Ultra-450 f</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Ultra-30/60 g</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Ultra-5/10 h</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

The audio mixer's mode may be changed at any time using the mixerctl command.

Audio Mixer Mode

The configuration file /usr/kernel/drv/audiocs.conf or /usr/kernel/drv/sparcv9/audiocs.conf is used to configure the audiocs driver so that the audio mixer is enabled or disabled. See the mixer(7I) manual page for details. The audio mixer’s mode may be changed at any time using the mixerctl command.

Audio Data Formats

The audiocs device supports the audio formats listed in the following table. When the audio mixer is in compatibility mode and the device is open for simultaneous play and record, the input and output data formats must match. Some sample rates are supported in compatibility mode that aren't supported in mixer mode. This is due to the computational overhead for sample rate conversion being too high.
<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>5510 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>6620 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>27420 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>33075 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>mu-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>5510 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>6620 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
</tbody>
</table>
### Sample Granularity

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Bits</th>
<th>Channels</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>18900 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>27420 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>C only</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>33075 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
</tbody>
</table>

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.

### Interrupt Rate

The driver determines how often play and record interrupts should happen. For playing audio this determines how often and how much audio is requested from the audio mixer. The impact on recording is minimal, however, if a very small read buffer size is set then the interrupt rate should be increased. The play and record interrupt rates are tuneable in the `audiocs.conf` file.

### 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.

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

### FILES

/`dev/audio`
- Symlink to the system’s primary audio device, not necessarily an `audiocs` audio device.

/`dev/audiocntl`
- Control device for the above audio device.

/`dev/sound/0`
- Represents the first audio device on the system and is not necessarily an `audiocs` audio device.

/`dev/sound/0ctl`
- Audio control for above device.

/`usr/demo/SOUND`
- Audio demonstration programs and other files.

/`usr/kernel/drv/audiocs`
- `audiocs` driver.
/usr/kernel/drv/audiocs.conf
    audiocs driver configuration file.

/usr/kernel/drv/sparcv9/audiocs
    audiocs driver, 64-bit.

/usr/kernel/drv/sparcv9/audiocs.conf
    audiocs 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, PowerPC on Solaris 2.5.1 only</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWaudd, SUNWauddx</td>
</tr>
<tr>
<td>Stability Level</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

mixerctl(1), ioctl(2), attributes(5), audio(7I), mixer(7I), streamio(7I)

Crystal Semiconductor, Inc., data sheet for the CS4231
# The audio support driver

The audio support driver supports audio drivers that use the new audio driver audio driver architecture. It also provides a limited number of ioctl(2)s for application programmers.

## Data Structures

The following data structures are defined to manage the different audio devices types and their channels.

### Device Types

The following enumeration lists a number of generic device types.

```c
typedef enum {
    UNDEFINED, AUDIO, AUDIOCTL, USER1, USER2, USER3
} audio_device_type_e;
```

At this time, Solaris implements only the AUDIO and AUDIOCTL audio device types, using the audio mixer, see mixer(7I) for details. The USER1, USER2, and USER3 device types allow third parties to write audio personality modules of their own.

### Channel Structure

This structure is used to get and set state information on individual channels.

```c
struct audio_channel {
    pid_t    pid; /* application’s process ID */
    uint_t   ch_number; /* channel this device is using */
    audio_device_type_e    dev_type; /* the device type */
    uint_t    info_size; /* size of the channel’s info structure */
    void     *info;    /* the channel’s state information */
} audio_channel_t;
```

The `ch_number` must be set to the specific channel number to get or set. When the `ioctl()` returns the `pid` will contain the process ID of the process that has the channel open and `dev_type` will contain the type of the device. If `pid` is 0 (zero), then the channel is not open. The pointer `info` must point to a buffer large enough to hold whatever audio device related state structure may be returned. At this time there is only the `audio_info_t` structure, see the audio(7I) and mixer(7I) man pages.

### IOCTLs

The audio support driver provides the following `ioctl()`s:

- **AUDIO_GET_CH_NUMBER**
  - This `ioctl()` returns the channel number the file descriptor represents in the integer pointed to by the `ioctl()` argument.

- **AUDIO_GET_CH_TYPE**
  - This `ioctl()` returns the type of channel the process has open via the `audio_device_type_e` enumeration pointed to by the `ioctl()` argument.

- **AUDIO_GET_NUM_CHS**
  - This `ioctl()` returns the number of channels the device supports in the integer

---

**NAME**

`audio_support` – audio driver support routines and interface

**SYNOPSIS**

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

**DESCRIPTION**

The audio support driver supports audio drivers that use the new audio driver audio driver architecture. It also provides a limited number of ioctl(2)s for application programmers.

**DATA STRUCTURES**

The following data structures are defined to manage the different audio devices types and their channels.

**Device Types**

The following enumeration lists a number of generic device types.

```c
typedef enum {
    UNDEFINED, AUDIO, AUDIOCTL, USER1, USER2, USER3
} audio_device_type_e;
```

At this time, Solaris implements only the AUDIO and AUDIOCTL audio device types, using the audio mixer, see mixer(7I) for details. The USER1, USER2, and USER3 device types allow third parties to write audio personality modules of their own.

**Channel Structure**

This structure is used to get and set state information on individual channels.

```c
struct audio_channel {
    pid_t    pid; /* application’s process ID */
    uint_t   ch_number; /* channel this device is using */
    audio_device_type_e    dev_type; /* the device type */
    uint_t    info_size; /* size of the channel’s info structure */
    void     *info;    /* the channel’s state information */
} audio_channel_t;
```

The `ch_number` must be set to the specific channel number to get or set. When the `ioctl()` returns the `pid` will contain the process ID of the process that has the channel open and `dev_type` will contain the type of the device. If `pid` is 0 (zero), then the channel is not open. The pointer `info` must point to a buffer large enough to hold whatever audio device related state structure may be returned. At this time there is only the `audio_info_t` structure, see the audio(7I) and mixer(7I) man pages.

**IOCTLs**

The audio support driver provides the following `ioctl()`s:

- **AUDIO_GET_CH_NUMBER**
  - This `ioctl()` returns the channel number the file descriptor represents in the integer pointed to by the `ioctl()` argument.

- **AUDIO_GET_CH_TYPE**
  - This `ioctl()` returns the type of channel the process has open via the `audio_device_type_e` enumeration pointed to by the `ioctl()` argument.

- **AUDIO_GET_NUM_CHS**
  - This `ioctl()` returns the number of channels the device supports in the integer

---

**Device and Network Interfaces**

49
pointed to by the ioctl() argument.

The following macro can be used to initialize data structures. The established convention is that the state corresponding to a field set to -1 will not be modified.

```
AUDIO_INIT(I, S)
```

Where I is a pointer to an info structure and S is the size of that structure.

The following code segment demonstrates how to use this macro:

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

**ERRORS**

- **EINVAL** The ioctl() is invalid for this file descriptor, the audio_channel_t structure's info pointer doesn't point to a buffer or the ch_number is bad.
- **ENOMEM** The ioctl() failed due to lack of memory.

**FILES**

`/usr/demo/SOUND` audio demonstration programs and other files

**ATTRIBUTES**

See attributes(5) for a description 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>
<tr>
<td>Availability</td>
<td>SUNWaudd, SUNWauddx, SUNWaudh</td>
</tr>
<tr>
<td>Stability Level</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

ioctl(2), open(2), audio(7I) mixer(7I) streamio(7I)

**FUTURE DIRECTIONS**

Over time additional audio personality modules will be added. The audio application programmer is encouraged to review this man page on each Solaris release for new audio personality modules.
The audiots device uses the ALI M5451 audio processor and an AC-97 Codec to implement the audio device interface.

This interface is described in the mixer(7I) and audio(7I) man pages.

Applications that open /dev/audio may use the AUDIO_GETDEV ioctl to determine which audio device is being used. The audiots driver will return the string SUNW,audiots in the name field of the audio_device structure. The version field will contain a letter (defined in the table below) and the config field will contain the string onboard1.

<table>
<thead>
<tr>
<th>Platform Type</th>
<th>Version</th>
<th>Line Out</th>
<th>Headphone</th>
<th>Int. Spkr</th>
<th>Line In</th>
<th>Mic</th>
<th>CD-ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grover</td>
<td>a</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

The audiots device provides support for line out, headphone, internal speaker, line in, and microphone. The play.mod_ports and record.mod_ports fields indicate which ports may be manipulated.

The configuration file /usr/kernel/drv/audiots.conf is used to configure the audiots driver so that the audio mixer is enabled or disabled. See the mixer(7I) manual page for details. The audio mixer mode may be changed at any time using the mixerctl(1) command.

The audiots device supports the audio formats listed in the following table. When the audio mixer is disabled and the device is opened for simultaneous play and record, the input and output data formats may be different. Some sample rates are supported when the mixer is disabled (D) that are not supported when the mixer is enabled (E), due to the overly high computational overhead for sample rate conversion.

The following table describes all supported data formats.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>5510 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>6620 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>Encoding</td>
<td>Precision</td>
<td>Channels</td>
<td>Mode</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-----------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>27420 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>33075 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>µ-Law/ A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>5510 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>6620 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>27420 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>D only</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>33075 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
<td>E and D</td>
</tr>
</tbody>
</table>

Because the audiots device manipulates buffers of audio data, the reported input and output sample counts will vary, at any given time, from the actual sample count by no more than the size of the buffers the audiots driver is transferring. In general, programs should not rely on the absolute accuracy of the play.samples and record.samples fields of the audio_info structure.
The driver determines how often play and record interrupts should take place. For playing audio, this determines how often and how much audio is requested from the audio mixer. The impact on recording is minimal, however, if a very small read buffer size is set. The play and record interrupt rates are tuneable in the 
/usr/kernel/drv/audiots.conf file.

As described in the audio(7I) and mixer(7I) man pages, it is possible to request asynchronous notification of changes in the state of an audio device.

audiots errors are described in the audio(7I) man page.

/dev/audio
Symbolic link to the system’s primary audio device. (Not necessarily an audiots audio device).
/dev/audioctl
Control device for the primary audio device.
/dev/sound/0
Represents the first audio device on the system. (Not necessarily an audiots audio device).
/dev/sound/0ctl
Audio control for /dev/sound/0.
/usr/demo/SOUND
Audio demonstration programs and other files.
/usr/kernel/drv/audiots.conf
audiots driver configuration file
/usr/kernel/drv/sparcv9/audiots
64-bit audiots driver

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

init_state() play interrupt rate set too low
The interrupt rate set in audiots.conf is set too low. It is being reset to the rate specified in the message. Update audiots.conf to a higher interrupt rate.

init_state() play interrupt rate set too high
The interrupt rate set in audiots.conf is set too high. It is being reset to the rate specified in the message. Update audiots.conf to a lower interrupt rate.

init_state() record interrupt rate set too low
The interrupt rate set in audiots.conf is set too low. It is being reset to the rate specified in the message. Update audiots.conf to a higher interrupt rate.

init_state() record interrupt rate set too high
The interrupt rate set in audiots.conf is set too high. It is being reset to the rate specified in the message. Update audiots.conf to a lower interrupt rate.
See attributes(5) for a 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>
<tr>
<td>Availability</td>
<td>SUNWaudd, SUNWauddx</td>
</tr>
</tbody>
</table>

SEE ALSO mixctl(1), ioctl(2), attributes(5), audio(7I), mixer(7I), streamio(7I)
Acer Laboratories Inc. M5451 PCI Audio Processor Technical Specification
authmd5h(7M)

NAME
authmd5h – HMAC-MD5 Authentication Algorithm Module for IPsec

SYNOPSIS
strmod/authmd5h

DESCRIPTION
This module implements the HMAC-MD5 authentication algorithm using the MD5 message-digest algorithm and the HMAC technique documented in RFC 2104. The authmd5h module has the following properties:

- key size 128 bits
- digest size 96 bits (truncated from 128)

authmd5h is used by both AH and ESP.

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>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO
ipseckey(1M), attributes(5), pf_key(7P), ipsec(7P), ipsecah(7P), ipsecesp(7P)


authsha1(7M)

NAME
authsha1 – HMAC-SHA-1 Authentication Algorithm Module for IPsec

SYNOPSIS
strmod/authsha1

DESCRIPTION
This module implements the HMAC-SHA-1 authentication algorithm, using the SHA-1 hash algorithm and the HMAC technique set forth in RFC 2104. The authsha1 module has the following properties:

key size 160 bits
digest size 96 bits (truncated from 160). authsha1 is used by both AH and ESP.

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>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO
ipseckey(1M), attributes(5), pf_key(7P), ipsec(7P), ipsecah(7P), ipsecesp(7P),
open("/dev/bd", O_RDWR)

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 ("VUID") 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(s) (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 VUID ioctl(s).

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.
bd(7M)

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_button(6), x_dial(6), streamIo(7I),
termic(7I)

SunButtons Installation and Programmers Guide
SunDials Installation and Programmers Guide

WARNINGS
The SunDials dial box must be used with a serial port.
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.

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

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.

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.

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.

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

The following ioctl requests are supported:

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

**BPPIOC_GETPARMS**

Get current transfer parameters.

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

**BPPIOC_SETOUTPINS**

Set output pin values.

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

**BPPIOC_GETOUTPINS**

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

**BPPIOC_GETERR**

Get last error status.

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

**BPPIOC_TESTIO**

Test transfer readiness.

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

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
        * read_handshake only!
        */
    BPP_XSCAN_HS, /* xerox scanner mode,
        * read_handshake only!
        */
    BPP_HSCAN_HS, /*
        * HP scanjet scanner mode
        * read_handshake only!
        */
    BPP_CLEAR_MEM, /* write 0’s to memory,
        * read_handshake only!
        */
    BPP_SET_MEM, /* write 1’s to memory,
        * read_handshake only!
        */
    /* The following handshakes are RESERVED. Do not use. */
    BPP_VPRINT_HS, /* valid only in read/write mode */
    BPP_VPLOT_HS /* valid only in read/write mode */
};
```

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

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

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

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

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

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

```c
struct bpp_pins {
    uchar_t output_reg_pins; /* pins in P_OR register */
    uchar_t input_reg_pins; /* pins in P_IR register */
};
/* Values for output_reg_pins field */
#define BPP_SLCTIN_PIN 0x01 /* Select in pin */
#define BPP_AFX_PIN 0x02 /* Auto feed pin */
#define BPP_INIT_PIN 0x04 /* Initialize pin */
#define BPP_V1_PIN 0x08 /* reserved pin 1 */
#define BPP_V2_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 */
```

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**
A BPPIOC_SETPARMS ioctl is attempted with an out of range value in the bpp_transfer_parms structure. A BPPIOC_SETOUTPINS ioctl is attempted with an invalid value in the pins structure. An ioctl is attempted with an invalid value in the command argument. An invalid command argument is received during modload(1M) or modunload(1M).

**EIO**
The driver encountered an SBus bus error when attempting an access.

A read or write does not complete properly, due to a peripheral error or a transfer timeout.

A BPPIOC_TESTIO ioctl call is attempted while a condition exists which would prevent a transfer (such as a peripheral error).

**ENXIO**
The driver has received an open request for a unit for which the attach failed. The driver has received a read or write request for a unit number greater than the number of units available. The driver has received a write request for a unit which has an active peripheral error.

**FILES**
/dev/bppn  bi-directional parallel port devices

**SEE ALSO**
ioctl(2), read(2), write(2), sbus(4)
bufmod – STREAMS Buffer Module

NAME

SYNOPSIS

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.

The behavior of bufmod depends on various parameters and flags that can be set and queried as described below under IOCTLs. Bufmod collects incoming M_DATA messages into chunks, passing each chunk upstream when the chunk becomes full or the current read timeout expires. It optionally converts M_PROTO messages to M_DATA and adds them to chunks as well. It also optionally adds to each message a header containing a timestamp, and a cumulative count of messages dropped on the stream read side due to resource exhaustion or flow control. The default settings of bufmod allow it to drop messages when flow control sets in or resources are exhausted; disabling headers and explicitly requesting no drops makes bufmod pass all messages through. Finally, bufmod is capable of truncating upstream messages to a fixed, programmable length.

When a message arrives, bufmod processes it in several steps. The following paragraphs discuss each step in turn.

Upon receiving a message from below, if the SB_NO_HEADER flag is not set, bufmod immediately timestamps it and saves the current time value for later insertion in the header described below.

Next, if SB_NO_PROTO_CVT is not set, bufmod converts all leading M_PROTO blocks in the message to M_DATA blocks, altering only the message type field and leaving the contents alone.

It then truncates the message to the current snapshot length, which is set with the SBIOCSSNAP ioctl described below.

Afterwards, if SB_NO_HEADER is not set, bufmod prepends a header to the converted message. This header is defined as follows.

```c
struct sb_hdr {
    uint_t sbh_origlen;
    uint_t sbh_msglen;
    uint_t sbh_totlen;
    uint_t sbh_drops;
    #if defined(_LP64) || defined(_I32LPx)
        struct timeval32 sbh_timestamp;
    #else
        struct timeval sbh_timestamp;
    #endif /* !_LP64 */
    #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 **SB_DEFER_CHUNK** flag is set, buffering does not begin until the second message is received within the timeout window.

If the **SB_SEND_ON_WRITE** flag is set, **bufmod** passes up the read side any buffered data when a message is received on the write side. **SB_SEND_ON_WRITE** and **SB_DEFER_CHUNK** are often used together.

**bufmod** intercepts **M_IOCTL** messages for the **ioctl()**s described below. The module passes all other messages through unaltered to its upper neighbor. If
**SB_SEND_ON_WRITE** is set, message arrival on the writer side suffices to close and transmit the current read side chunk.

**IOCTLS**

bufmod responds to the following ioctl.

<table>
<thead>
<tr>
<th>_IOCTL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIOCSTIME</td>
<td>Set the read timeout value to the value referred to by the struct timeval pointer given as argument. Setting the timeout value to zero has the side-effect of forcing the chunk size to zero as well, so that the module will pass all incoming messages upward immediately upon arrival. Negative values are rejected with an EINVAL error.</td>
</tr>
<tr>
<td>SBIOCGTIME</td>
<td>Return the read timeout in the struct timeval pointed to by the argument. If the timeout has been cleared with the SBIOCSTIME ioctl, return with an ERANGE error.</td>
</tr>
<tr>
<td>SBIOCCTIME</td>
<td>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.</td>
</tr>
<tr>
<td>SBIOCSCHUNK</td>
<td>Set the chunk size to the value referred to by the uint_t pointer given as argument. See NOTES for a description of effect on stream head high water mark.</td>
</tr>
<tr>
<td>SBIOCGCHUNK</td>
<td>Return the chunk size in the uint_t pointed to by the argument.</td>
</tr>
<tr>
<td>SBIOCSSNAP</td>
<td>Set the current snapshot length to the value given in the uint_t pointed to by the ioctl’s final argument. bufmod interprets a snapshot length value of zero as meaning infinity, so it will not alter the message. See NOTES for a description of effect on stream head high water mark.</td>
</tr>
<tr>
<td>SBIOCGSNAP</td>
<td>Returns the current snapshot length in the uint_t pointed to by the ioctl’s final argument.</td>
</tr>
<tr>
<td>SBIOCSFLAGS</td>
<td>Set the current flags to the value given in the uint_t pointed to by the ioctl’s final argument. Possible values are a combination of the following.</td>
</tr>
</tbody>
</table>

- **SB_SEND_ON_WRITE** Transmit the read side chunk on arrival of a message on the write side.
- **SB_NO_HEADER** Do not add headers to read side messages.
- **SB_NO_DROPS** Do not drop messages due to flow control upstream.
- **SB_NO_PROTO_CVT** Do not convert M_PROTO messages into M_DATA.
SB_DEFER_CHUNK  Begin buffering on arrival of the second read side message in a timeout interval.

SBIOCGFLAGS  Returns the current flags in the uint_t pointed to by the ioctl's final argument.

SEE ALSO  dlpi(7P), le(7D), pfmod(7M)

NOTES  Older versions of bufmod did not support the behavioral flexibility controlled by the SBIOLSFLAGS 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 SBIOLSCHUNK 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.
bwtwo(7D)

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.
The cadp160 host bus adapter driver is a SCSA-compliant nexus driver that supports the following Adaptec Ultra160 SCSI devices:

- Adapters: 39160, 29160, 29160N, 29160LP

The cadp160 driver supports standard functions provided by the SCSA interface including tagged and untagged queuing, wide, fast and ultra SCSI, and auto request sense. The cadp160 driver does not support linked commands. The cadp160 driver supports hot swap SCSI, hot plug PCI, 64-bit addressing (dual address cycle), domain validation, PCI bus clock rates up to 66MHz and narrow and wide devices at 20MB/sec, 40MB/sec, 80MB/sec, and 160MB/sec.

FILES

/platform/i86pc/kernel/drv/cadp160
ELF kernel module

/boot/solaris/drivers/notisa.010/cadp160.bef
Realmode BEF driver

/platform/i86pc/kernel/drv/cadp160.conf
Optional configuration file

ATTRIBUTES

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

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

SEE ALSO

pvtconf(1M), driver.conf(4), pci(4), attributes(5), dlpi(7P),
scsi_abort(9F), scsi_hba_attach(9F), scsi_ifgetcap(9F),
scsi_ifsetcap(9F), scsi_reset(9F), scsi_sync_pkt(9F),
scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S),
scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

Solaris (Intel Platform Edition) Hardware Compatibility List

ANSI Small Computer System Interface-2 (SCSI-2)
The `cadp` host bus adapter driver is a SCSA–compliant nexus driver that supports the following Adaptec Ultra-2 SCSI Devices:

- Chips: AIC-7860, AIC-7880, AIC-7895, AIC-7896/AIC-7897, AIC-7890/AIC-7891, AIC-7890AB, AIC-7890A

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

Driver Configuration

The `cadp` host bus adapter driver is configured by defining the properties found in `cadp.conf`. Properties in the `cadp.conf` file that can be modified by the user include: `scsi-options`, `target<n>-scsi-options`, `scsi-reset-delay`, and `scsi-initiator-id`. Properties in the `cadp.conf` file override global SCSI settings.

The property `target<n>-scsi-options` overrides the `scsi-options` property value for `target<n>`, where `<n>` can vary from decimal 0 to 15. The `cadp` driver supports the following `scsi-options`: `SCSI_OPTIONS_DR`, `SCSI_OPTIONS_SYNC`, `SCSI_OPTIONS_TAG`, `SCSI_OPTIONS_FAST`, `SCSI_OPTIONS_WIDE`, `SCSI_OPTIONS_FAST20`, and `SCSI_OPTIONS_FAST40`.

EXAMPLES

EXAMPLE 1

Create a file called `/kernel/drv/cadp.conf`, then add the following line:

```
scsi-options=0x78;
```

The above line disables tagged queuing, Fast/Ultra SCSI, and wide mode for all `cadp` instances.

To set `scsi-options` more specifically per target, add the following lines to `/kernel/drv/cadp.conf`:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
  "SEAGATE ST32550W", "seagate-scsi-options";
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

With the exception of one disk type that has `scsi-options` set to `0x58`, the above example sets `scsi-options` for target 1 to `0x78`, and all remaining targets to `0x3f8`. 
EXAMPLE 1 (Continued)

The `scsi-options` properties that are specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` for all `cadp` instances per bus have the lowest precedence. You must reboot the system for the specified `scsi options` to take effect.

### Driver Capabilities

To enable certain features on the `cadp` driver, the target driver must set capabilities. The following capabilities can be queried and modified by the target driver: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, and `qfull-retry-interval`. All other capabilities are query only.

By default, the `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled. The `disconnect`, `synchronous`, and `untagged-qing` capabilities are always enabled. The `cadp` driver capabilities can only be assigned binary values (0 or 1). The default value for `qfull-retries` is 10 and the default value for `qfull-retry-interval` is 100. The `qfull-retries` capability is `au_char` (0 to 255) while `qfull-retry-interval` is a `u_short` (0 to 65535).

If a conflict occurs between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call. See `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

### FILES

`/kernel/drv/cadp` ELF kernel module

`/kernel/drv/cadp.conf` Optional configuration file

### ATTRIBUTES

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

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

### SEE ALSO

`prtconf(1M)`, `driver.conf(4)`, `pci(4)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

*Hardware Compatibility List for Solaris 8 (Intel Platform Edition)*

*ANSI Small Computer System Interface-2 (SCSI-2)*
The \texttt{cadp} driver supports the adapters and chipsets listed in this man page. For information on support of additional devices, see the \texttt{Hardware Compatibility List for Solaris 8 (Intel Platform Edition)}, a component of the \texttt{Information Library for Solaris 8 (Intel Platform Edition)}. The \texttt{cadp} driver exports properties indicating (per target) the negotiated transfer speed (\texttt{target<n>-sync-speed}), whether wide bus (\texttt{target<n>-wide}), is supported for that particular target (\texttt{target<n>-scsi-options}), and whether tagged queuing (\texttt{target<n>-tag-queue}) has been enabled. The \texttt{sync-speed} property value is the data transfer rate in KB/sec. The \texttt{target<n>-tag-queue} and the \texttt{target<n>-wide} property have value 1 to indicate that the corresponding capability is enabled, or 0 to indicate that the capability is disabled. See \texttt{prtconf(1M)} (verbose option) for information on viewing the \texttt{cadp} properties.

Sample output is provided below:

\begin{verbatim}
pci9005,f500, instance #2
System software properties:
  name <interrupt-priorities> length <4>
  value <0x05000000>.
  name <tape> length <5>
  value <0x7363747000>.
  name <disk> length <5>
  value <0x7363646b00>.
  name <queue> length <6>
  value <0x71736f727400>.
  name <flow_control> length <6>
  value <0x646d756c7400>.

Driver properties:
  name <target0-tag-queue> length <4>
  value <0x01000000>.
  name <target0-wide> length <4>
  value <0x01000000>.
  name <target0-sync-speed> length <4>
  value <0x28000000>.
  name <chosen-interrupt> length <8>
  value <0x0100000000000000>.
  name <scsi-selection-timeout> length <4>
  value <0xf0000000>.
  name <scsi-options> length <4>
  value <0xf81f0000>.
  name <scsi-watchdog-tick> length <4>
  value <0x0a000000>.
  name <scsi-tag-age-limit> length <4>
  value <0x02000000>.
  name <scsi-reset-delay> length <4>
  value <0xb80b0000>.
\end{verbatim}
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.

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.

```c
/*
 * definition of play audio msf structure
 */
struct cdrom_msf {
    unsigned char cdmsf_min0; /* starting minute*/
    unsigned char cdmsf_sec0; /* starting second*/
    unsigned char cdmsf_frame0; /* starting frame*/
    unsigned char cdmsf_min1; /* ending minute */
```
unsigned char cdmsf_sec1; /* ending second */
unsigned char cdmsf_frame1; /* ending frame */

The **CDROMREADTOCENTRY** ioctl request may be used to obtain the start time for a track. An approximation of the finish time can be obtained by using the **CDROMREADTOCENTRY** ioctl request to retrieve the start time of the track following the current track.

The leadout track is the next consecutive track after the last audio track. Hence, the start time of the leadout track may be used as the effective finish time of the last audio track.

**CDROMPLAYTRKIND**

This ioctl() command is similar to **CDROMPLAYMSF**. The starting and ending address is in track/index format. The third argument of the ioctl() call is a pointer to the type struct cdrom_ti.

/*
 * definition of play audio track/index structure
 */
struct cdrom_ti {
  unsigned char cdti_trk0; /* starting track*/
  unsigned char cdti_ind0; /* starting index*/
  unsigned char cdti_trk1; /* ending track */
  unsigned char cdti_ind1; /* ending index */
};

**CDROMVOLCTRL**

This ioctl() command controls the audio output level. The SCSI command allows the control of up to four channels. The current implementation of the supported CD-ROM drive only uses channel 0 and channel 1. The valid values of volume control are between 0x00 and 0xFF, with a value of 0xFF indicating maximum volume. The third argument of the ioctl() call is a pointer to struct cdrom_volctrl which contains the output volume values.

/*
 * definition of audio volume control structure
 */
struct cdrom_volctrl {
  unsigned char channel0;
  unsigned char channel1;
  unsigned char channel2;
  unsigned char channel3;
};

The following I/O controls take a pointer that will have data returned to the user program from the CD-ROM driver.
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.

```c
/*
 * 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
CDROM_SUBCHNL
This ioctl() command reads the Q sub-channel data of the current block. The subchannel data includes track number, index number, absolute CD-ROM address, track relative CD-ROM address, control data and audio status. All information is returned through a pointer to struct cdrom_subchnl. The caller needs to supply the address format for the returned address.

```
struct cdrom_subchnl {
    unsigned char cdsc_format;
    unsigned char cdsc_audiostatus;
    unsigned char cdsc_adr: 4;
    unsigned char cdsc_ctrl: 4;
    unsigned char cdsc_trk;
    unsigned char cdsc_ind;
    union {
        struct {
            unsigned char minute;
            unsigned char second;
            unsigned char frame;
        } msf;
        int lba;
    } cdsc_absaddr;
    union {
        struct {
            unsigned char minute;
            unsigned char second;
            unsigned char frame;
        } msf;
        int lba;
    } cdsc_reladdr;
};
```

The following values are valid for the audio status field returned from READ_SUBCHANNEL command:

- CDROM_AUDIO_INVALID Audio status not supported.
- CDROM_AUDIO_PLAY Audio play operation in progress.
- CDROM_AUDIO_PAUSED Audio play operation paused.
- CDROM_AUDIO_COMPLETED Audio play successfully completed.
- CDROM_AUDIO_ERROR Audio play stopped due to error.
- CDROM_AUDIO_NO_STATUS No current audio status to return.

CDROMREADOFFSET
This ioctl() command returns the absolute CD-ROM address of the first track in the last session of a Multi-Session CD-ROM. The third argument of the ioctl() call is a pointer to an int.
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().

```c
/*
 * 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.
- 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

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;
};
```
To get the proper CD-ROMXA data, the following values are appropriate for the cdxa_format field:

- CDROM_XA_DATA: CD-ROMXA data only
- CDROM_XA_SECTOR_DATA: CD-ROMXA all sector data
- CDROM_XA_DATA_W_ERROR: CD-ROMXA data with error flags data

To allocate the memory related to CD-ROMXA format, the following values are appropriate for each data block transferred:

- CD-ROMXA data only: 2048 bytes
- CD-ROMXA all sector data: 2352 bytes
- CD-ROMXA 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.

CDROMGBLKMODE

This ioctl() command returns the current block size used by the CD-ROM drive. The third argument of the ioctl() call is a pointer to an integer.

CDROMSBLKMODE

This ioctl() command requests the CD-ROM drive to change from the current block size to the requested block size. The third argument of the ioctl() call is an integer which contains the requested block size.
This `ioctl()` command operates in exclusive-use mode only. The caller must ensure that no other processes can operate on the same CD-ROM device before issuing this `ioctl()`. The `read(2)` behavior subsequent to this `ioctl()` remains the same: the caller is still constrained to read the raw device on block boundaries and in block multiples.

To set the proper block size, the following values are appropriate:

- **CDROM_BLK_512**: 512 bytes
- **CDROM_BLK_1024**: 1024 bytes
- **CDROM_BLK_2048**: 2048 bytes
- **CDROM_BLK_2056**: 2056 bytes
- **CDROM_BLK_2336**: 2336 bytes
- **CDROM_BLK_2340**: 2340 bytes
- **CDROM_BLK_2352**: 2352 bytes
- **CDROM_BLK_2368**: 2368 bytes
- **CDROM_BLK_2448**: 2448 bytes
- **CDROM_BLK_2646**: 2646 bytes
- **CDROM_BLK_2647**: 2647 bytes

**CDROMGDRVSPEED**

This `ioctl()` command returns the current CD-ROM drive speed. The third argument of the `ioctl()` call is a pointer to an integer.

**CDROMSDRVSPEED**

This `ioctl()` command requests the CD-ROM drive to change the current drive speed to the requested drive speed. This speed setting is only applicable when reading data areas. The third argument of the `ioctl()` is an integer which contains the requested drive speed.

To set the CD-ROM drive to the proper speed, the following values are appropriate:

- **CDROM_NORMAL_SPEED**: 150k/second
- **CDROM_DOUBLE_SPEED**: 300k/second
- **CDROM_QUAD_SPEED**: 600k/second
- **CDROM_MAXIMUM_SPEED**: 300k/second (2x drive) 600k/second (4x drive)

Note that these numbers are only accurate when reading 2048 byte blocks. The CD-ROM drive will automatically switch to normal speed when playing audio tracks and will switch back to the speed setting when accessing data.

**SEE ALSO**

`ioctl(2), read(2)`
 NOTES  

The CDROMCDDA, CDROMCDXA, CDROMSUBCODE, CDROMGDRVSPEED, CDROMSDRVSPED, 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.
ce – Cassini Gigabit-Ethernet device driver
/dev/ce

DESCRIPTION
The ce Sun Gigabit-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over all implementations of PCI Cassini Gigabit-Ethernet add-in adapters. Multiple Cassini-based adapters installed within the system are supported by the driver. The ce driver provides basic support for the Cassini-based Ethernet hardware and handles the pci108e,abba (PCI Cassini) devices. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting. The Cassini device provides 1000BASE-SX networking interfaces using the Cassini ASIC external SERDES and fiber optical transceiver, or 10/100/1000BASE-T using a Cassini ASIC attached to a GMII twisted pair copper transceiver, or 10/100BASE-T using a Cassini ASIC attached to a MII twisted pair copper transceiver.

The 1000Base-SX standard specifies an auto-negotiation protocol to automatically select the mode of operation. In addition to the duplex mode of operation, the Cassini ASIC can auto-negotiate for IEEE 802.3x frame-based flow control capabilities. The Cassini PCS can perform auto-negotiation with the link’s remote-end (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. It also supports forced-mode of operation where the driver selects the mode of operation.

The /dev/ce cloning character-special device is used to access all ce controllers installed on the system.

The ce driver is a Style 2 data link service provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in <sys/dlpi.h>. Refer to dlpi(7P) for more information. An explicit DL_ATTACH_REQ message by the user is required to associate the opened stream with a particular device (ppa). The ppa ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. An error (DL_ERROR_ACK) is returned by the driver if the ppa field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) upon last detach.

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

- Maximum SDU is 1500 (ETHERMTU - defined in <sys/ethernet.h>).
- Minimum SDU is 0.
- The dlsap address length is 8.
- MAC type is DL_ETHER.
- The sap length value is –2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
Service mode is DL_CLDLS.
Optional quality of service (QOS) is not supported; the QOS fields are 0.
Provider style is DL_STYLE2.
Version is DL_VERSION_2.
Broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFFFF)

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular SAP (Service Access Pointer) with the stream. The ce driver interprets the sap field within the DL_BIND_REQ as an Ethernet “type,” therefore valid values for the sap field are in the range [0-0xFFFF]. Only one Ethernet type can be bound to the stream at any time.

If you select a sap with a value of 0, the receiver will be in 802.3 mode. All frames received from the media having a “type” field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open streams which are bound to sap value 0. If more than one stream is in 802.3 mode, the frame will be duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ to verify that the sap value is 0, and that the destination type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The ce driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should not hard code to this particular implementation-specific DLSAP address format, but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, you can transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the ce driver. The ce driver will route received Ethernet frames up all open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams, if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory connectionless DLPI message set, the driver additionally supports the following primitives.

**ce Primitives**
The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be
iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS option set in the dl_level field enables/disables reception of all "promiscuous mode" frames on the media, including frames generated by the local host. When used with the DL_PROMISC_SAP option set, this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI option set this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive because it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

**ce DRIVER**

By default, the ce driver performs auto-negotiation to select the mode and flow control capabilities of the link.

The link can assume one of the following modes:

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- Symmetric pause
- Asymmetric pause

Speeds and modes are described in the 1000Base-TX standard.

The auto-negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Flow control capability (symmetric and/or asymmetric)

The auto-negotiation protocol does the following:

- Gets all modes of operation supported by the link partner.
- Advertises its capabilities to the link partner.
Selects the highest common denominator mode of operation based on the priorities.

The Cassini hardware can operate in all modes listed above, providing auto-negotiation is used by default to bring up the link and select the common mode of operation with the link partner. The PCS also supports forced-mode of operation in which the driver can select the mode of operation and the flow control capabilities, using the ndd utility.

The Cassini device also supports programmable IPG (Inter-Packet Gap) parameters ipg1 and ipg2. By default, the driver sets ipg1 and ipg2 to 8 and 4 byte-times respectively (which are the standard values). If desired, you can alter these values from the standard 1000 Mbps IPG set to 0.096 microseconds.

The ce driver enables the setting and getting of various parameters for the Cassini device. The parameter list includes current transceiver status, current link status, inter-packet gap, PCS capabilities and link partner capabilities.

The PCS features two set of capabilities. One set reflects the capabilities of the hardware and are read-only. The second set, which reflects the values you choose, are used in speed selection and possess read/write capabilities. At boot time, these two sets of capabilities are the same. The link partner capabilities are also read-only because the current default value of these parameters can be read but not modified.

**FILES**

| /dev/ce | ce special character device. |
| /kernel/drv/ce.conf | System-wide default device driver properties |

**SEE ALSO**

ndd(1M), netstat(1M), driver.conf(4), ge(7D), hme(7D), le(7D), qfe(7D), dlpi(7P)
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 Plane 1152x900x1
0x250000 -> 24-bit Frame Buffer 1152x900x32
```

FILES
/dev/fbs/cgeight

<sys/fbio.h>

<sys/pr_planegroups.h>

SEE ALSO
mmap(2), fbio(7I)

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 Plane 1152x900x1
0x250000 -> 24-bit Frame Buffer 1152x900x32
```

FILES
/dev/fbs/cgeight

<sys/fbio.h>

<sys/pr_planegroups.h>

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

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:

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

The cgfourteen device can serve as a system console device.

See `/usr/include/sys/cg14io.h` for other device-specific information.

### FILES

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/kernel/drv/cgfourteen</code></td>
<td>cgfourteen device driver</td>
</tr>
<tr>
<td><code>/dev/fbs/cgfourteen[n0-9]</code></td>
<td>Logical device name.</td>
</tr>
<tr>
<td><code>/usr/include/sys/cg14io.h</code></td>
<td>Header file that contains device specific information</td>
</tr>
<tr>
<td><code>/usr/include/sys/cg14reg.h</code></td>
<td>Header file that contains device specific information</td>
</tr>
</tbody>
</table>

### SEE ALSO

`mmap(2)`, `fbio(7I)`
cgsix(7D)

NAME
cgsix – accelerated 8-bit color frame buffer

SYNOPSIS
/dev/fbs/cgsix

DESCRIPTION
cgsix is a low-end graphics accelerator designed to enhance vector and polygon 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)
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/cn[dn|p]n block device (IDE)
/dev/rdsk/cn[dn|p]n raw device (IDE)

where:
  cn   controller n
  dn   lun n (0-7)
  sn   UNIX system slice n (0-15)
  pn   fdisk partition (0)

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>IA</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)
The **cnft** Ethernet driver is a multi-threaded, loadable, clonable, STREAMS GLD driver. This driver supports the following controllers:

- Compaq NetFlex-3/EISA
  - 10Base-T UTP Module
  - 10/100Base-TX UTP Module
  - 100VG-AnyLAN UTP Module
  - 100Base-FX Module
- Compaq NetFlex-3/PCI
  - 10Base-T UTP Module
  - 10/100Base-TX UTP Module
  - 100VG-AnyLAN UTP Module
  - 100Base-FX Module
- Compaq Netelligent 10Base-T PCI UTP
- Compaq Netelligent 10/100 TX PCI UTP
- Compaq Dual Port NetFlex-3 10/100TX PCI UTP
- Compaq Integrated NetFlex-3 10/100T PCI with AUI on ProLiant 2500 and Professional Workstation 5000
- Compaq Integrated NIC on DeskPro 4000/6000 and ProLiant 800

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

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

```
example% 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

```
example% 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.
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.

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

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

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

This property is used to support additional controller IDs. The format is 0xVIDDID where VID is the Vendor ID and DID the device ID.

/dev/cnft
/kernel/drv/cnft.conf
/sys/stropts.h
/sys/ethernet.h
/sys/gld.h

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>IA</td>
</tr>
</tbody>
</table>
SEE ALSO attributes(5), dlpi(7P)
connld – line discipline for unique stream connections

/dev/connld

connld is a STREAMS-based module that provides unique connections between server and client processes. It can only be pushed (see streamio(7I)) onto one end of a STREAMS-based pipe that may subsequently be attached to a name in the file system name space with fattach(3C). After the pipe end is attached, a new pipe is created internally when an originating process attempts to open(2) or creat(2) the file system name. A file descriptor for one end of the new pipe is packaged into a message identical to that for the ioctl I_SENDFD (see streamio(7I)) and is transmitted along the stream to the server process on the other end. The originating process is blocked until the server responds.

The server responds to the I_SENDFD request by accepting the file descriptor through the I_RECVFD ioctl message. When this happens, the file descriptor associated with the other end of the new pipe is transmitted to the originating process as the file descriptor returned from open(2) or creat(2).

If the server does not respond to the I_SENDFD request, the stream that the connld module is pushed on becomes uni-directional because the server will not be able to retrieve any data off the stream until the I_RECVFD request is issued. If the server process exits before issuing the I_RECVFD request, the open(2) or the creat(2) invocation will fail and return -1 to the originating process.

When the connld module is pushed onto a pipe, it ignores messages going back and forth through the pipe.

ERRORS

On success, an open of connld returns 0. On failure, errno is set to the following values:

EINVAL
A stream onto which connld is being pushed is not a pipe or the pipe does not have a write queue pointer pointing to a stream head read queue.

EINVAL
The other end of the pipe onto which connld is being pushed is linked under a multiplexor.

EPIPE
connld is being pushed onto a pipe end whose other end is no longer there.

ENOMEM
An internal pipe could not be created.

ENXIO
An M_HANGUP message is at the stream head of the pipe onto which connld is being pushed.

EAGAIN
Internal data structures could not be allocated.

ENFILE
A file table entry could not be allocated.

SEE ALSO
creat(2), open(2), fattach(3C), streamio(7I)
console(7D)

NAME  console – STREAMS-based console interface
SYNOPSIS  /dev/console
DESCRIPTION  The file /dev/console refers to the system console device. /dev/console should be used for interactive purposes only. Use of /dev/console for logging purposes is discouraged; syslog(3C) or msglog(7D) should be used instead.

The identity of this device depends on the EEPROM or NVRAM settings in effect at the most recent system reboot; by default, it is the “workstation console” device consisting of the workstation keyboard and frame buffer acting in concert to emulate an ASCII terminal (see wscons(7D)).

Regardless of the system configuration, the console device provides asynchronous serial driver semantics so that, in conjunction with the STREAMS line discipline module ldterm(7M), it supports the termio(7I) terminal interface.

SEE ALSO  syslog(3C), termios(3C), ldterm(7M), termio(7I), msglog(7D), wscons(7D)
NOTES  In contrast to pre-SunOS 5.0 releases, it is no longer possible to redirect I/O intended for /dev/console to some other device. Instead, redirection now applies to the workstation console device using a revised programming interface (see wscons(7D)). Since the system console is normally configured to be the 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.
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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO

driver.conf(4), attributes(5)
### NAME
cpr - Suspend and resume module

### SYNOPSIS
/platform/`uname -m`/kernel/misc/cpr

### DESCRIPTION
The `cpr` module is a loadable module used to suspend and resume the entire system. You may wish to suspend a system to save power or to power off temporarily for transport. The `cpr` module should not be used in place of a normal shutdown when performing any hardware reconfiguration or replacement. In order for the resume operation to succeed, it is important that the hardware configuration remain the same. When the system is suspended, the entire system state is preserved in non-volatile storage until a resume operation is conducted.

dtpower(1M) or `power.conf(4)` are used to configure the suspend-resume feature.

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

During resume operation, the `SIGTHAW` signal is sent to all processes to allow them to do any special processing in response to suspend-resume operation. Normally applications are not required to do any special processing because of suspend-resume, but some specialized processes can use `SIGTHAW` to restore the state prior to suspend. For example, X can refresh the screen in response to `SIGTHAW`.

In some cases the `cpr` module may be unable to perform the suspend operation. If a system contains additional devices outside the standard shipped configuration, it is possible that device drivers for these additional devices might not support suspend-resume operations. In this case, the suspend will fail and an error message will be displayed. These devices must be removed or their device drivers unloaded for the suspend operation to succeed. Contact the device manufacturer to obtain a new version of device driver that supports suspend-resume.

A suspend may also fail when devices or processes are performing critical or time-sensitive operations (such as realtime operations). The system will remain in its current running state. Messages reporting the failure will be displayed on the console and status returned to the caller. Once the system is successfully suspended the resume operation will 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.

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

dtpower(1M) (OpenWindows Reference Manual), pmconfig(1M), uadmin(1M), uadmin(2), power.conf(4), attributes(5)

Using Power Management

Writing Device Drivers

NOTES

Certain device operations such as tape and floppy disk activities are not resumable due to the nature of removable media. These activities are detected at suspend time, and must be stopped before the suspend operation will complete successfully.

Suspend-resume is currently supported only on a limited set of hardware platforms. Please see the book Using Power Management for a complete list of platforms that support system Power Management. See uname(2) to programmatically determine if the machine supports suspend-resume.

BUGS

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 such that the system does not resume and 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  cvc – virtual console driver

DESCRIPTION  The cvc virtual console driver is a STREAMS-based pseudo driver that supports the network console. The cvc driver interfaces with console(7D).

Logically, the cvc driver sits below the console driver. It redirects console output to the cvcredir(7D) driver if a network console connection is active. If a network console connection is not active, it redirects console output to an internal hardware interface.

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

NOTES  The cvc facility supersedes the SunOS wscons(7D) facility, which should not be used in conjunction with cvc. The wscons driver is useful for systems with directly attached consoles (frame buffers and keyboards), but is not useful with platforms using cvc, which have no local keyboard or frame buffer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Sun Enterprise 10000 servers, Sun Fire 15000 servers</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)

*Sun Enterprise 10000 SSP Reference Manual*

*Sun System Management Services (SMS) Reference Manual*
cvcredir(7D)

NAME
cvcredir – virtual console redirection driver

DESCRIPTION The cvcredir virtual console redirection driver is a STREAMS-based pseudo driver that supports the network console provided on some platforms. The cvcredir driver interfaces with the virtual console driver cvc(7D), and the virtual console daemon, cvcd(1M).

The cvcredir driver receives console output from cvc and passes it to cvcd. It receives console input from cvcd and passes it to cvc.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Sun Enterprise 10000 servers, Sun Fire 15000 servers</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcvc.u</td>
</tr>
</tbody>
</table>

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

Sun Enterprise 10000 SSP Reference Manual

Sun System Management Services (SMS) Reference Manual
NAME  
dad – driver for IDE disk devices

SYNOPSIS  
dad@ target, lun: partition

DESCRIPTION  
This driver handles the ide disk drives on SPARC platforms.

The type of disk drive is determined using the ATA IDE identify device command and by reading the volume label stored on 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 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.

Device Statistics Support

Each device maintains I/O statistics both for the device and for each partition allocated on that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also takes hi-resolution time stamps at queue entry and exit points, which facilitates monitoring the residence time and cumulative residence-length product for each queue.

Each device also has error statistics associated with it. These must include counters for hard errors, soft errors and transport errors. Other data may be implemented as required.

FILES

/dev/dsk/cntndnsn  block files
/dev/rdsk/cntndnsn  raw files

where:

\( cn \)  controller \( n \)
\( tn \)  IDE target id \( n \) (0-3)
\( dn \)  Always 0.
\( sn \)  partition \( n \) (0-7)

The target ide numbers are assigned as:

0  Master disk on Primary channel.
1  Slave disk on Primary channel.
Master disk on Secondary channel
Slave disk on Secondary channel.

**IOCTLs**
Refer to `dkio(7I)`.

**ERRORS**
- **EACCES**
  Permission denied.
- **EBUSY**
  The partition was opened exclusively by another thread.
- **EFAULT**
  The argument was a bad address.
- **EINVAL**
  Invalid argument.
- **EIO**
  An I/O error occurred.
- **ENOTTY**
  This indicates that the device does not support the requested ioctl function.
- **ENXIO**
  During opening, the device did not exist.
- **EROFS**
  The device is a read-only device.

**SEE ALSO**
- `format(1M)`, `mount(1M)`, `lseek(2)`, `read(2)`, `write(2)`, `driver.conf(4)`, `vfstab(4)`, `dkio(7I)`
- X3T10 ATA-4 specifications.

**DIAGNOSTICS**
- **offline**
  The driver has decided that the target disk is no longer there.
- **disk ok**
  The target disk is now responding again.
- **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.
- **disk not responding to selection**
  The target disk is not responding.
- **i/o to invalid geometry**
  The geometry of the drive could not be established.
- **incomplete read/write - retrying/giving up**
  There was a residue after the command completed normally.
- **no bp for disk label**
  A bp with consistent memory could not be allocated.
- **no memory for disk label**
  Free memory pool exhausted.
ATA transport failed: reason ‘nnnn’: {retrying|giving}
   The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

corrupt label - wrong magic number
   The disk label is corrupted.

corrupt label - label checksum failed
   The disk label is corrupted.

corrupt label - bad geometry
   The disk label is corrupted.

no mem for property
   Free memory pool exhausted.

transport rejected (<n>)
   Host adapter driver was unable to accept a command.

Device Fault
   There has been a Device Fault - reason for such error is vendor specific.
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(7I)` 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.
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 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>

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(7)` 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>
<tr>
<td>HDLC</td>
<td>16000 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>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>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
/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
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.

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.

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)

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(3DEVINFO), attributes(5)

Writing Device Drivers
dkio(7)  

NAME    dkio -- disk control operations

SYNOPSIS  
#include <sys/dkio.h>
#include <sys/vtoc.h>

DESCRIPTION  
Disk drivers support a set of ioctl(2) requests for disk controller, geometry, and partition information. Basic to these ioctl() requests are the definitions in <sys/dkio.h>.

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

DKIOCINFO  
The argument is a pointer to a dk_cinfo structure (described below). This structure tells the controller-type and attributes regarding bad-block processing done on the controller.

/*
 * Structures and definitions for disk I/O control commands
 */
#define DK_DEVLEN 16 /* device name max length, */
    /* including unit # and NULL */
    /* Used for controller info */
struct dk_cinfo {
    char    dki_cname[DK_DEVLEN];  /* controller name */
        /*(no unit #)*/
    ushort_t dki_ctype;  /* controller type */
    ushort_t dki_flags;  /* flags */
    ushort_t dki_cnum;  /* controller number */
    uint_t    dki_addr;  /* controller address */
    uint_t    dki_space;  /* controller bus type */
    uint_t    dki_prio;  /* interrupt priority */
    uint_t    dki_vec;  /* interrupt vector */
    char    dki_dname[DK_DEVLEN];  /* drive name (no unit #) */
    uint_t    dki_unit;  /* unit number */
    uint_t    dki_slave;  /* slave number */
    ushort_t    dki_partition;  /* partition number */
    ushort_t    dki_maxtransfer;  /* maximum transfer size */
        /* in DEV_BSIZE */
};

/*
 * Controller types
 */
#define DKC_UNKNOWN 0
#define DKC_CDROM 1 /* CD-ROM, SCSI or other */
#define DKC_WDC2880 2
#define DKC_XXX_0 3 /* unassigned */
#define DKC_XXX_1 4 /* unassigned */
#define DKC_DSD5215 5
#define DKC_ACB4000 7
#define DKC_SMSFLOPPY 12
#define DKC_SCSI_CCS 13 /* SCSI CCS compatible */

114  man pages section 7: Device and Network Interfaces • Last Revised 19 April 2001
#define DKC_INTEL82072 14 /* native floppy chip */
#define DKC_MD 16 /* meta-disk (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 */

/* driver */

# define DKC_CUSTOMER_BASE 1024

/ *
 * Sun reserves up through 1023
 */

#define DKI_BAD144 0x01 /* use DEC std 144 */
#define DKI_MAPTRK 0x02 /* controller does */
#define DKI_FMTTRK 0x04 /* formats only full */
#define DKI_FMTVOL 0x08 /* formats only full */
#define DKI_FMTCYL 0x10 /* formats only full */
#define DKI_HEXUNIT 0x20 /* unit number printed */
#define DKI_PCMCIA_PFD 0x40 /* PCMCIA pseudo-floppy */

/ *
 * Sun reserves up through 1023
 */

#define DKC_CUSTOMER_BASE 1024

/ *
 * Flags
 */

#define DKI_BAD144 0x01 /* use DEC std 144 */
#define DKI_MAPTRK 0x02 /* controller does */
#define DKI_FMTTRK 0x04 /* formats only full */
#define DKI_FMTVOL 0x08 /* formats only full */
#define DKI_FMTCYL 0x10 /* formats only full */
#define DKI_HEXUNIT 0x20 /* unit number printed */
#define DKI_PCMCIA_PFD 0x40 /* PCMCIA pseudo-floppy */

/ *
 * Sun reserves up through 1023
 */

#define DKC_CUSTOMER_BASE 1024

/ *
 * Flags
 */

#define DKI_BAD144 0x01 /* use DEC std 144 */
#define DKI_MAPTRK 0x02 /* controller does */
#define DKI_FMTTRK 0x04 /* formats only full */
#define DKI_FMTVOL 0x08 /* formats only full */
#define DKI_FMTCYL 0x10 /* formats only full */
#define DKI_HEXUNIT 0x20 /* unit number printed */
#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.

```c
/* 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_map {
  struct dk_allmap {
    struct dk_map dka_map[NDKMAP];
  };
*/

**DKIOCGGEOM**  The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` gets the controller’s notion of the current geometry of the disk drive.

**DKIOCSGEOM**  The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` sets the controller’s notion of the geometry without changing the disk itself.

**DKIOCGVTOC**  The argument is a pointer to a `vtoc` structure (described below). This `ioctl()` returns the device’s current volume table of contents (VTOC.)

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

```c
struct partition {
    ushort_t p_tag;  /* ID tag of partition */
    ushort_t p_flag; /* permission flags */
    daddr_t p_start; /* start sector of partition */
    long p_size;    /* # of blocks in partition */
};
```

If **DKIOCSVTOC** is used with a floppy diskette, the `p_start` field must be the first sector of a cylinder. To compute the number of sectors per cylinder, multiply the number of heads by the number of sectors per track.

```c
struct vtoc {
    unsigned long v_bootinfo[3];  /* info needed by mboot */
};
```
unsigned long v_sanity; /* to verify vtoc sanity */
unsigned long v_version; /* layout version */
char v_volume[LEN_DKL_VVOL]; /* volume name */
ushort_t v_sectorsz; /* sector size in bytes*/
ushort_t v_nparts; /* number of partitions*/
unsigned long v_reserved[10]; /* free space */
struct partition v_part[V_NUMPAR]; /* partition headers*/
time_t timestamp[V_NUMPAR]; /* partition timestamp */
char v_asciilabel[LEN_DKL_ASCII]; /* compatibility */
}

/* Partition permission flags */
#define V_UNMNT 0x01 /* Unmountable partition */
#define V_RONLY 0x10 /* Read only */

/* Partition identification tags */
#define V_UNASSIGNED 0x00 /* unassigned partition */
#define V_BOOT 0x01 /* Boot partition */
#define V_ROOT 0x02 /* Root filesystem */
#define V_SWAP 0x03 /* Swap filesystem */
#define V_USR 0x04 /* Usr filesystem */
#define V_BACKUP 0x05 /* full disk */
#define V_VAR 0x07 /* Var partition */
#define V_HOME 0x08 /* Home partition */
#define V_ALTSCTR 0x09 /* Alternate sector partition */

DKIOCEJECT
If the drive supports removable media, this ioctl() requests the disk drive to eject its disk.

DKIOCREMOVABLE
The argument to this ioctl() is an integer. After successful completion, this ioctl() 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.

DKIOCSTATE
This ioctl() blocks until the state of the drive, inserted or ejected, is changed. The argument is a pointer to a dkio_state, enum, whose possible enumerations are listed below. The initial value should be either the last reported state of the drive, or DKIO_NONE. Upon return, the enum pointed to by the argument is updated with the current state of the drive.
enum dkio_state {
    DKIO_NONE,    /* Return disk's current state */
    DKIO_EJECTED, /* Disk state is 'ejected' */
    DKIO_INSERTED /* Disk state is 'inserted' */
};

DKIOCLOCK For devices with removable media, this ioctl() requests the disk drive to lock the door.

DKIOCUNLOCK For devices with removable media, this ioctl() requests the disk drive to unlock the door.

DKIOCGMEDIAINFO The argument to this ioctl() is a pointer to a dk_minfo structure. The structure indicates the type of media or the command set profile used by the drive to operate on the media. The dk_minfo structure also indicates the logical media blocksize the drive uses as the basic unit blocksize of operation and the raw formatted capacity of the media in number of logical blocks.

/*
 * Used for media info or profile info
 */
struct dk_minfo {
    uint_t dki_media_type; /* Media type or profile info */
    uint_t dki_lbsize;    /* Logical blocksize of media */
    diskaddr_t dki_capacity; /* Capacity as # of dki_lbsize blks */
};
/*
 * Media types or profiles known
 */
#define DK_UNKNOWN 0x00 /* Media inserted - type unknown */

/*
 * SFF 8090 Specification Version 3, media types 0x01 - 0xfffe are retained to
 * maintain compatibility with SFF8090. The following define the
 * optical media type.
 */
#define DK_MO_ERASABLE 0x03 /* MO Erasable */
#define DK_MO_WRITEONCE 0x04 /* MO Write once */
#define DK_AS_MO 0x05 /* AS MO */
#define DK_CDROM 0x08 /* CDROM */
#define DK_CDR 0x09 /* CD-R */
#define DK_CDRW 0x0A /* CD-RW */
#define DK_DVDR 0x10 /* DVD-ROM */
#define DK_DVDRAM 0x12 /* DVD_RAM or DVD-RW */

/*
 * Media types for other rewritable magnetic media
 */
#define DK_FIXED_DISK 0x10001 /* Fixed disk SCSI or otherwise */
#define DK_FLOPPY 0x10002 /* Floppy media */
#define DK_ZIP 0x10003 /* IOMEGA ZIP media */
#define DK_JAZ 0x10004 /* IOMEGA JAZ media */

If the media exists and the host can obtain a current profile list, the command will succeed and return the dk_minfo structure with data representing that media.

If there is no media in the drive, the command will fail and the host will return an ENXIO error, indicating that it cannot gather the information requested.

If the profile list is not available, the host will attempt to identify the media-type based on the available information.

If identification is not possible, the host will return media type DK_UNKNOWN. See NOTES for blocksize usage and capacity information.

**DKIOCSMBOOT**

The argument is a pointer to struct mboot.

Copies the mboot information supplied in the argument to the absolute sector 0 of the device. Prior to copying the information, this ioctl() performs the following checks on the mboot data:

- Ensures that the signature field is set to 0xAA55.
- Ensures that partitions do not overlap.
- On SPARC platforms, determines if the device is a removable media. If the above verification fails, errno will be set to EINVAL and the ioctl() command will fail.

**IA Platforms** — Upon successful write of mboot, the partition map structure maintained in the driver is updated. If the new Solaris partition is different from the previous one, the internal VTOC table maintained in the driver will be set as follows:

If _SUNOS_VTOC_8 is defined:

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>START</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Capacity of device</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Capacity of device</td>
</tr>
</tbody>
</table>

If _SUNOS_VTOC_16 is defined:
To determine if the Solaris partition has changed:

If either offset or the size of the Solaris partition is different from the previous one then it shall be deemed to have changed. In all other cases, the internal VTOC info will remain as before.

SPARC Platforms — The VTOC label and mboot both occupy the same location, namely sector 0. As a result, following the successful write of mboot info, the internal VTOC table maintained in the driver will be set as follows:

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>START</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Capacity of device</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Capacity of device</td>
</tr>
</tbody>
</table>

See the NOTES section for usage of DKIOMSBMBOOT when modifying Solaris partitions.

DKIOMSBMBOOT
The argument is a pointer to struct mboot. The 512 bytes of absolute sector 0 of the device is copied to the mboot structure pointed to by the argument.

RETURN VALUES
Upon successful completion, the value returned is 0. Otherwise, -1 is returned and errno is set to indicate the error.

IA Only
The following ioctl() requests set and/or retrieve the current disk controller, partitions, or geometry information on IA architecture.

DKIOCG_PHYGEOM
The argument is a pointer to a dk_geom structure (described below). This ioctl() gets the driver’s notion of the physical geometry of the disk drive. It is functionally identical to the DKIOCGGEOM ioctl().
DKIOCG_VIRTGEOM  The argument is a pointer to a \texttt{dk\_geom} structure (described below). This \texttt{ioctl()} gets the controller’s (and hence the driver’s) notion of the virtual geometry of the disk drive. Virtual geometry is a view of the disk geometry maintained by the firmware in a host bus adapter or disk controller. If the disk is larger than 8 Gbytes, this \texttt{ioctl} will fail because a CHS-based geometry is not relevant or useful for this drive.

/*
 * 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\_intrlv; /* interleave factor */
unsigned short  dkg\_obs2; /* obsolete */
unsigned short  dkg\_obs3; /* obsolete */
unsigned short  dkg\_apc; /* alternates per cylinder */
(unsigned short  dkg\_rpm; /* revolutions per min*/
(unsigned short  dkg\_pcyl; /* # of physical cylinders */
(unsigned short  dkg\_write\_reinstruct; /* # sectors to skip, writes*/
(unsigned short  dkg\_read\_reinstruct; /* # sectors to skip, reads*/
(unsigned short  dkg\_extra[7]; /* for compatible expansion*/

#define dkg\_gap1 dkg\_extra[0] /* for application compatibility*/
#define dkg\_gap2 dkg\_extra[1] /* for application compatibility*/

DKIOCADDBAD  This \texttt{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. \texttt{DKIOCADDBAD} can only be used for software remapping on IDE drives; SCSI drives use hardware remapping of alternate sectors.

DKIOCPRINFO  The argument is a pointer to a \texttt{part\_info} structure (described below). This \texttt{ioctl()} gets the driver’s notion of the size and extent of the partition or slice indicated by the file descriptor argument.

/*
 * Used by applications to get partition or slice information
 */

/\struct\ part\_info

daddr\_t   p\_start;
int         p\_length;

*/
Blocksize information provided in \texttt{DKIOCGMEDIAINFO} is the size (in bytes) of the device’s basic unit of operation and may differ from the blocksize that the Solaris operating environment exports to the user. Capacity information provided in the \texttt{DKIOCGMEDIAINFO} are for reference only and you are advised to use the values returned by \texttt{DKIOCGGEOOM} or other appropriate \texttt{ioctl} for accessing data using the standard interfaces.

IA only: If the \texttt{DKIOCSMBOOT} command is used to modify the Solaris partitions, the VTOC information should also be set appropriately to reflect the changes to partition. Failure to do so will lead to unexpected results when the device is closed and re-opened fresh at a later time. This is because a default VTOC is assumed by driver when a Solaris partition is changed. The default VTOC will persist until the ioctl \texttt{DKIOCSVTOC} is called to modify VTOC or the device is closed and re-opened. At that point, the old valid VTOC will be read from the disk if it is still available.
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**

`1e(7D)`
The dman(7D) network device driver is a loadable, clonable, STREAMS hardware driver that supports the connectionless data link provider interface dlpi(7P) over the SUNW, dman network controller. The dman controller provides a highly available, secure communication channel between the dynamic system domains and the Sun Fire 15000 system controller.

The dman driver provides basic support for the SUNW, dman controller. Driver functions include network initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The dman controller is physically located in each Sun Fire 15000 dynamic system domain and connects to SUNW,sman controllers in the Sun Fire 15000 chassis. See scman(7D). All links are point-to-point and are internal to the Sun Fire 15000 chassis. Traffic between the dynamic system domains and the system controller is not accessible by any third party; for example, another system domain within the Sun Fire 15000 chassis. Only the system controller is accessible through the dman controller.

The link layer frame format is identical to that used by Ethernet (sys/ethernet.h).

The /dev/dman cloning character-special device is used to access the SUNW, dman controller installed on the system.

The dman driver is a style 2 data link provider interface. 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 or physical point of attachment (PPA).

The PPA ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. The only valid unit number is 0. 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 the first attach and deinitialized (stopped) upon the last detach.

The values returned by the dman driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are:

- Maximum service data units (SDU) are 1500.
- Minimum SDU is 0.
- Data link service access point (DLSAP) address length is 8.
- Media access control (MAC) type is DL_ETHER.
- Service access point (SAP) length value is -2, meaning the physical address component is followed immediately by a two-byte SAP component within the DLSAP address.
Service mode is DL_CLDLs.
Optional quality of service (QOS) support is not included; the QOS fields are 0.
Provider style is DL_STYLE2.
Version is DL_VERSION_2.
Broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFFF).

Once in the DL_ATTACHED state, you can transmit a DL_BIND_REQ to associate a particular SAP with the stream. The dman driver interprets the SAP field within the DL_BIND_REQ as an Ethernet type; as a result, valid values for the SAP field are in the 0 through 0xFFFF range. Only one Ethernet type can be bound to the stream at any time. If you select the 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 from 0 through 1500 are assumed to be 802.3 frames and are routed up all open streams that are bound to SAP value 0. If more than one stream is in 802.3 mode, the frame is duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the DL_BIND_REQ SAP field to determine if the SAP value is 0 and the destination type field is in the range from 0 through 1500. If either is true, the driver computes the length of the message, not including the initial M_PROTO message block (mblk), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The dman driver DLSAP address format consists of the six-byte physical (Ethernet) address component followed immediately by the two-byte SAP (type) component producing an eight-byte DLSAP address. Applications should not be hard-coded to this implementation-specific DLSAP address format, but 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, you can transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the dman driver. The dman driver routes received Ethernet frames as DL_UNITDATA_IND messages up all open and bound streams having a SAP matching the Ethernet type. 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.

PRIMITIVES

In addition to the mandatory connectionless DLPI message set, the dman driver supports the following 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 other SAP and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the six-octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive is not supported.

dman DRIVER

The dman driver operates at 10 Mbps, full-duplex.

PARAMETER LIST

The dman driver allows you to set and get various parameters for the SUNW, dman device. The parameter list includes current transceiver status, current link status, interpacket gap, local transceiver capabilities, and link partner capabilities.

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 Fire 15000 servers</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcar</td>
</tr>
</tbody>
</table>

FILES

The dman driver utilizes the following files:

/dev/dman
Special character device

/platform/SUNW,Sun-Fire-15000/kernel/drv/dman.conf
System-wide default device driver properties

SEE ALSO

ndd(1M), netstat(1M), driver.conf(4), eri(7D), scman(7D), dlpi(7P)
The dmfe Ethernet device provides 100Base-TX networking interfaces using the Davicom DM9102A chip, which incorporates its own internal transceiver.

The dmfe driver is a multithreaded, loadable, clonable, GLD-based STREAMS driver. Multiple controllers installed within the system are supported by the driver. The dmfe driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

The 100Base-TX standard specifies an auto-negotiation protocol to automatically select the mode and speed of operation. The internal transceiver is capable of performing autonegotiation with the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. The internal transceiver also supports a forced-mode of operation under which the driver selects the operational mode.

The /dev/dmfe cloning character-special device is used to access all Davicom DM9102A devices installed in the system.

The dmfe driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the dmfe 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.

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

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

- Maximum SDU is 1500 (ETHERMTU - defined in sys/ethernet.h).
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is DL_ETHER.
- The sap length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- The broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.
dmfe(7D)

CONFUGURATION

By default, the dmfe driver performs auto-negotiation to select the speed and mode of the link. Link speed and mode can be 100 Mbps (full or half-duplex) or 10 Mbps (full or half-duplex) as described in the 100Base-TX standard.

The auto-negotiation protocol automatically selects speed mode (either 100 Mbps or 10 Mbps) and operation mode (either full-duplex or half-duplex) as the highest common denominator supported by both link partners. Because the dmfe device supports all modes, this effectively selects the highest-throughput mode supported by the other device.

Alternatively, you can explicitly specify the link parameters by adding entries to the dmfe driver configuration file (/platform/SUNW,UltraAX-i2/kernel/drv/dmfe.conf). You can set the speed parameter to 10 or 100 to force dmfe devices to operate at the specified speed. Additionally, you can set the full-duplex parameter to 0 or 1 to disable or force full-duplex operation, respectively.

Note that specifying either "speed" or "full-duplex" explicitly disables auto-negotiation. To enable the driver to determine the appropriate setting for each parameter, you should always set both parameters. If it is necessary to force either speed or duplex setting (for example, because the dmfe device is connected to an ancient device or hub that does not support auto-negotiation), both parameters should be explicitly specified to match the requirements of the external device.

FILES

/dev/dmfe
  Character special device

/platform/SUNW,UltraAX-i2/kernel/drv/dmfe.conf
  dmfe 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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO

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

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide
**NAME**

dnet – Ethernet driver for DEC 21040, 21041, 21140 Ethernet cards

**SYNOPSIS**

/kernel/drv/dnet

**DESCRIPTION**

The dnet Ethernet driver is a multithreaded, loadable, clonable, STREAMS GLD driver. Multiple controllers installed within the system are supported by the driver. The dnet driver functions include controller initialization, frame transmit and receive, functional addresses, promiscuous and multicast support, and error recovery and reporting.

The cloning character-special device, /dev/dnet, is used to access all DEC 21040/21041/21140 devices installed in the system.

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
dnet(7D)

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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO

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

*Writing Device Drivers*

*STREAMS Programming Guide*

*Network Interface Guide*
The dynamic reconfiguration (DR) driver consists of a platform-independent driver and a platform-specific module. The DR driver uses standard features of the Solaris operating environment whenever possible to control DR operations and calls the platform specific module as needed. The DR driver creates minor nodes in the file system that serve as attachment points for DR operations.

The DR driver provides a pseudo-driver interface to sequence attach and detach operations on system boards using file system entry points or "attachment points." The form of the attachment point depends on the DR model running on the domain.

**DR 2.0**

The DR 2.0 model uses `dr_daemon(1M)` to control DR operations and runs only on Sun Enterprise 10000 servers. For this server, the DR driver consists of a platform-independent driver (`dr`) and a platform-specific module (`drmach`).

The `drmach(7d)` module provides platform-specific DR functions and attributes on the Sun Enterprise 10000 server. The DR daemon, `dr_daemon(1M)`, executes the DR operations by making the appropriate `ioctl(2)` system calls into the respective attachment point for that particular board.

For DR 2.0, an attachment point exists for each possible system board slot and takes one of the following forms:

```
/devices/pseudo/dr@0:slotx
/devices/pseudo/dr@0:IOx
```

Where `x` represents the physical slot number (0 to 15) for a particular system board.

**DR 3.0**

The DR 3.0 model uses the domain configuration server, `dcs(1M)`, to control remote DR operations and runs on the Sun Enterprise 10000 and Sun Fire 15K servers.

**Sun Enterprise 10000 Server**

On the Sun Enterprise 10000 server, the DR driver consists of a platform-independent drive (`ngdr`) and a platform specific module (`ngdrmach`).

The domain configuration server (DCS) accepts DR requests from the system services processor (SSP) and uses the `libcfgadm(3LIB)` interface to initiate the DR operation. After the operation is performed, the results are returned to the SSP. For more information about the DCS on the Sun Enterprise 10000, refer to the `dcs(1M)` man page and the *Sun Enterprise 10000 Dynamic Reconfiguration User Guide*. 

---

**NAME**

`dr`, `drmach`, `ngdr`, `ngdrmach` – Sun Enterprise 10000 and Sun Fire 15K dynamic reconfiguration driver

**SYNOPSIS**

```
dr

  drmach

  ngdr

  ngdrmach
```

**DESCRIPTION**

The dynamic reconfiguration (DR) driver consists of a platform-independent driver and a platform-specific module. The DR driver uses standard features of the Solaris operating environment whenever possible to control DR operations and calls the platform specific module as needed. The DR driver creates minor nodes in the file system that serve as attachment points for DR operations.

The DR driver provides a pseudo-driver interface to sequence attach and detach operations on system boards using file system entry points or "attachment points." The form of the attachment point depends on the DR model running on the domain.
The DR driver creates a physical attachment point for system board slots that takes the following form:

```
/devices/pseudo/ngdr@0:SBx
```

Where \( x \) represents the slot number (0 to 15) for a particular board.

The `cfgadm_sbd(1M)` plugin creates dynamic attachment points. These attachment points refer to components on the system boards, including CPUs, memory, or I/O devices. Refer to the `cfgadm_sbd(1M)` man page for more details.

### Sun Fire 15K Server

On the Sun Fire 15K server, the DR driver consists of a platform-independent driver (`dr`) and a platform-specific module (`drmach`).

The domain configuration server (DCS) accepts DR requests from the domain configuration agent (DCA) that runs on the Sun Fire 15K system controller. After the DCS accepts a DR request, it uses the `libcfgadm(3LIB)` interface to initiate the DR operation. After the operation is performed, the results are returned to the DCA. For more information about the DCS, refer to the `dcs(1M)` man page and the *Sun Fire 15K Dynamic Reconfiguration User Guide*. For more information about the DCA, refer to the `dcs(1M)` man page and the *System Management Services (SMS) Dynamic Reconfiguration User Guide*.

The DR driver creates a physical attachment point for system board slots that take the following forms.

```
/devices/pseudo/dr@0:SBx
/devices/pseudo/dr@0:IOx
```

Where \( x \) represents the expander number (0 to 17) for a particular board.

The `cfgadm_sbd(1M)` plugin creates dynamic attachment points. These attachment points refer to components on the system boards, including CPUs, memory, or I/O devices. Refer to the `cfgadm_sbd(1M)` man page for more details.

### SEE ALSO

- `cfgadm_sbd(1M)`, `dcs(1M)` (available only on a Sun Fire system running SMS software), `dcs(1M)`, `dr_daemon(1M)`, `ioctl(2)`, `libcfgadm(3LIB)`
- *Sun Enterprise 10000 Dynamic Reconfiguration User Guide*
- *Sun Fire 15K Dynamic Reconfiguration User Guide*
- *System Management Services (SMS) Dynamic Reconfiguration User Guide*
ecpp – IEEE 1284 compliant parallel port driver

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

cpp@unit-address

Description

The ecpp driver provides a bi-directional interface to IEEE 1284 compliant devices as well as a forward single-directional interface to Centronics devices. In addition to the Centronics protocol, the ecpp driver supports the IEEE 1284 Compatibility, Nibble, and ECP protocols. ECPP_COMPAT_MODE and ECPP_CENTRONICS modes of operation have logically identical handshaking protocols, however devices that support ECPP_COMPAT_MODE are IEEE 1284 compliant devices. IEEE 1284 compliant devices support at least ECPP_COMPAT_MODE and ECPP_NIBBLE_MODE. Centronics devices support only ECPP_CENTRONICS mode.

By default, ECPP_COMPAT_MODE devices have a strobe handshaking pulse width of 500ns. For this mode, forward data transfers are conducted by DMA. By default, the strobe pulse width for ECPP_CENTRONICS devices is two microsecond. Forward transfers for these devices are managed through PIO. The default characteristics for both ECPP_COMPAT_MODE and ECPP_CENTRONICS devices may be changed through tunable variables defined in ecpp.conf.

The ecpp driver is an exclusive-use device, meaning that if the device has already been opened, subsequent opens fail with EBUSY.

Default Operation

Each time the ecpp device is opened, the device is marked as EBUSY and the configuration variables are set to their default values. The write_timeout period is set to 90 seconds.

The driver sets the mode variable according to the following algorithm: The driver initially attempts to negotiate the link into ECPP_ECP_MODE during open(2). If it fails, the driver will try to negotiate in ECPP_NIBBLE_MODE mode. If that fails, the driver will operate in ECPP_CENTRONICS mode. Upon successfully opening the device, IEEE 1284 compliant devices will be left idle in either reverse idle phase of ECPP_ECP_MODE or ECPP_NIBBLE_MODE. Subsequent calls to write(2) will invoke the driver to move the link into either forward of phase of ECPP_ECP_MODE or ECPP_COMPAT_MODE. After the transfer is complete, the link will return to idle state as described earlier.

The application may attempt to negotiate the device into a specific mode or set the write_timeout values through the ECPPIOC_SETPARMS ioctl(2) call. For mode negotiation to be successful, both the host workstation and the peripheral must support the requested mode.

Tunables

Characteristics of the ecpp driver may be tuned by the variables described in /kernel/drv/ecpp.conf. These variables are read while the driver is being
attached to the kernel. If the driver is currently attached, ecpp must be unloaded
before the driver can be re-attached and the tuneable variables read again. See
modunload (1M).

Some Centronics peripherals and certain IEEE 1284 compatible peripherals will not
operate with the parallel port operating in a fast handshaking mode. If printing
problems occur, set "fast-centronics" and "fast-1284-compatible" to "false." See
/kernel/drv/ecpp.conf for more information.

The ecpp driver is a full duplex STREAMS device driver. While an application is
writing to an IEEE 1284 compliant device, another thread may read from it.

A write(2) operation returns the number of bytes successfully written to the stream
head. If a failure occurs while a Centronics device is transferring data, the content of
the status bits will be captured at the time of the error, and can be retrieved by the
application program using the BPPIOC_GETERR ioctl(2) call. The captured status
information will be overwritten each time an attempted transfer or a BPPIOC_TESTIO
ioctl(2) occurs.

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(2) returns 0 if O_NDELAY is set. If O_NONBLOCK is
set, read(2) returns -1 and sets errno to EAGAIN. If O_NDELAY and O_NONBLOCK are
clear, read(2) blocks until data become available.

The ioctl(2) calls described below are supported. Note that if ecpp is currently
transferring data, the driver waits until the data has been sent to the device before
processing the ioctl(2) call.

ECPPIOC_GETPARMS
Get current transfer parameters. The argument is a pointer to a struct
ecpp_transfer_parms. See below for a description of the elements of this
structure. If no parameters have been configured since the device was opened, the
structure will be set to its default configuration. See Default Operation above for
more information.

ECPPIOC_SETPARMS
Set transfer parameters. The argument is a pointer to a struct
ecpp_transfer_parms. If a parameter is out of range, EINVAL is returned. If the
peripheral or host device cannot support the requested mode, EPROTONOSUPPORT
is returned. See below for a description of ecpp_transfer_parms and its valid
parameters.

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

struct ecpp_transfer_parms {
  int write_timeout;
  int mode;
};
The `write_timeout` field is set to the value of `ecpp-transfer-timeout` specified in the `ecpp.conf`. The `write_timeout` field specifies how long the driver will wait for the peripheral to respond to a transfer request. The value must be greater than 0 and less than `ECPP_MAX_TIMEOUT`. Any other values are out of range.

The `mode` field reflects the IEEE 1284 mode to which the parallel port is currently configured. The mode may be set to one of the following values only: `ECPP_CENTRONICS`, `ECPP_COMPAT_MODE`, `ECPP_NIBBLE_MODE`, `ECPP_ECP_MODE`. All other values are invalid. If the requested mode is not supported, `ECPPIOC_SETPARMS` will return `EPROTONOSUPPORT` and the mode will be set to `ECPP_CENTRONICS` mode. Afterwards, the application may change the mode back to the original mode with `ECPPIOC_SETPARMS`.

### ECPPIOC_GETDEVID

This ioctl gets the IEEE 1284 device ID from the peripheral in specified mode. Currently device ID can be retrieved only in Nibble mode. Pointer to the following structure defined in `<sys/ecppsys.h>` must be passed as an argument.

The 1284 device ID structure:

```c
struct ecpp_device_id {
    int mode; /* mode to use for reading device id */
    int len; /* length of buffer */
    int rlen; /* actual length of device id string */
    char *addr; /* buffer address */
};
```

The mode is the 1284 mode in which the port will be negotiated into to retrieve device ID information. If the peripheral or host do not support the mode, `EPROTONOSUPPORT` will be returned. Applications should set mode to `ECPP_NIBBLE_MODE`. `len` is the length of the buffer pointed to by `addr`. `rlen` is the actual length of the device ID string returned from the peripheral. If the returned `rlen` is greater than `len`, the application can call `ECPPIOC_GETDEVID` again with a buffer length equal or greater than `rlen`. Note that the two length bytes of the 1284 device ID are not taken into account and are not returned in the user buffer.

After `ECPPIOC_GETDEVID` successfully completes, the driver returns the link to `ECPP_COMPAT_MODE`. The application is responsible for determining the previous mode the link was operating in and returning the link to that mode.

### BPPIOC_TESTIO

Tests the forward transfer readiness of a peripheral operating in Centronics or Compatibility mode.

`TESTIO` determines if the peripheral is ready to receive data by checking the open flags and the Centronics status signals. If the current mode of the device is `ECPP_NIBBLE_MODE`, the driver will negotiate the link into `ECPP_COMPAT_MODE`, check the status signals and then return the link to `ECPP_NIBBLE_MODE` mode.
the current mode is ECPP_CENTRONICS or ECPP_COMPAT_MODE, TESTIO will examine the Centronics status signals in the current mode. To receive data, the device must have the nErr and Select signals asserted and must not have the PE and Busy signals asserted. If ecpp is currently transferring data, TESTIO waits until the previous data sent to the driver is delivered before executing TESTIO. However, if an error condition occurs while a TESTIO is waiting, TESTIO will return immediately. If TESTIO determines that the conditions are ok, 0 is returned. Otherwise, -1 is returned, errno is set to EIO, and the state of the status pins is captured. The captured status can be retrieved using the BPPIOC_GETERR ioctl(2) call. The timeout_occurred and bus_error fields will never be set by this ioctl(2). BPPIOC_TESTIO and BPPIOC_GETERR are compatible to the ioctls specified in bpp(7D).

BPPIOC_GETERR
Get last error status. The argument is a pointer to a struct bpp_error_status defined in <sys/bpp_io.h> header file. The error status structure is:

```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 */
};
```

The pin_status field indicates possible error conditions. The valid bits for pin_status are: BPP_ERR_ERR, BPP_SLCT_ERR, BPP_PE_ERR, BPP_BUSY_ERR. A set bit indicates that the associated pin is asserted.

This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a write(2) call, or the status of the bits at the most recent BPPIOC_TESTIO ioctl(2) call.

pin_status indicates possible error conditions under ECPP_CENTRONICS or ECPP_COMPAT_MODE. Under these modes, the state of the status pins will indicate the state of the device. For instance, many Centronics printers lower the nErr signal when a paper jam occurs. The behavior of the status pins depends on the device. Additional status information may be retrieved through the backchannel.

The timeout_occurred value is set when a timeout occurs during write(2). bus_error is not used in this interface.

The following ioctls are used to directly read and write the parallel port status and control signals. If the current mode of the device is ECPP_ECP_MODE or ECPP_NIBBLE_MODE, the driver will negotiate the link into ECPP_COMPAT_MODE, get or set the registers and then return the link to ECPP_NIBBLE_MODE. If the current mode is ECPP_CENTRONICS or ECPP_COMPAT_MODE, these ioctls will get/set the register values in the current mode.
**ECPPIOC_GETREGS**
Read register values. The argument is a pointer to a struct ecpp_regs. See below for a description of this structure.

**ECPPIOC_SETREGS**
Set ecpp register values. The argument is a pointer to a struct ecpp_regs. See below for a description of this structure. If a parameter is out of range, EINVAL is returned.

The Port Register Structure is defined in `<sys/ecpio.h>`.

```c
struct ecpp_regs {
    uchar dsr; /* status reg */
    u_char dcr; /* control reg */
};
```

The status register is read-only. The ECPPIOC_SETREGS ioctl has no affect on this register. Valid bit values for dsr are: ECPP_nER, ECPP_SLCT, ECPP_PE, ECPP_nACK, ECPP_nBUSY. All other bits are reserved and will always return 1.

The control resister is read/write. Valid bit values for dcr are: ECPP_STB, ECPP_AFX, ECPP_nINIT, ECPP_SLCTIN. All other bits are reserved. Reading reserved bits will always return 1. An attempt to write 0s into these bits will result in EINVAL.

**DEVICE SPECIAL FILES**
/dev/ecpp
/dev/printers

1284 compliant parallel port device special files appears in both namespaces.

**FILES**
kernel/drv/ecpp 32-bit ELF kernel module
kernel/drv/sparcv9/ecpp64-bit ELF kernel module
kernel/drv/ecpp.conf Driver configuration file

**ERRORS**
EBADF The device is opened for write-only access and a read is attempted, or the device is opened for read-only access and a write is attempted.

EBUSY The device has been opened and another open is attempted. An attempt has been made to unload the driver while one of the units is open.

EINVAL A ECPPIOC_SETPARMS ioctl() is attempted with an out-of-range value in the ecpp_transferParms structure. A ECPPIOC_SETREGS ioctl() is attempted with an invalid value in the ecpp_regs structure. An ioctl() is attempted with an invalid value in the command argument.An invalid command argument is received during modload(1M) or modunload(1M).

EIO The driver encountered a bus error when attempting an access. A read or write did not complete properly, due to a peripheral error or a transfer timeout.
The driver has received an open request for a unit for which the attach failed. The driver has received a write request for a unit which has an active peripheral error.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Limited to PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWpd</td>
</tr>
<tr>
<td>Interface stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

ioctl(2), read(2), write(2), system(4), bpp(7D), usbprn(7D), streamio(7I)

*IEEE Std 1284–1994*

**DIAGNOSTICS**

Parallel port controller not supported

Driver does not support parallel port controller on the given host. Attach failed.
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 IA 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 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

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>IA</td>
</tr>
</tbody>
</table>
SEE ALSO  attributes(5), dlpi(7P), gld(7D)
The `elxl` driver currently supports the following network cards: EtherLink XL (3C900-TPO, 3C900-COMBO, 3C900B-TPO, 3C900B-COMBO, and 3C900B-TPC), EtherLink XL 10/100 (3C905-TX Fast, 3C905-T4 Fast, 3C905B-TX Fast, 3C905B-T4 Fast, and 3C905C-TX-M Fast), and EtherLink Server 10/100 (3C980-TX Fast and 3C980C-TXM).

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

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

The `elxl` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `elxl` driver with the DLPI and STREAMS functionality required of a LAN driver. See `gld(7D)` for more details on the primitives supported by the driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- Maximum SDU is 1500 (ETHERMTU).
- Minimum SDU is 0. The driver will pad to the mandatory 60-octet minimum packet size.
- The `dlsap` address length is 8.
- MAC type is `DL_ETHER`.
- The `sap` length value is -2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the DLSAP address.
- The broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

**Supported Settings**

- Media Type: Auto Select

**Known Problems and Limitations**

- 3C905B cards in a Compaq ProLiant 6500 can fail to generate interrupts. There is no known workaround for this problem. However, because some slots appear to be more prone to the problem than others, try correcting the problem by moving the card to another PCI slot. If that fails, try rebooting the machine a number of times to free the card from the wedged state.

- Early versions of the 3Com 3C905C-TX-M adapter firmware do not support PXE network boot on Solaris systems. If you are using a version earlier than 4.11, upgrade the firmware. The PXE version is indicated by the Managed Boot Agent version number. This number is not normally displayed during boot, but is shown...
on the PXE configuration screen.

**CONFIGURATION**
The `/kernel/drv/elxl.conf` file supports the following option:

- `full-duplex` For full duplex operation use `full-duplex=1`. For half duplex use `full-duplex=0`. Half-duplex operation provides better results on older 10-Mbit networks.

**FILES**
- `/dev/elxl` Special character device
- `/kernel/drv/elxl.conf` Configuration file for elxl driver

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

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

**SEE ALSO** attributes(5), gld(7D), streamio(7I), dlpi(7P).
This module implements triple-DES, which is the application of the United States Data Encryption Standard ("DES") three times with three different keys for IPsec. The triple application of DES, given K1, K2, and K3, happens on a per-block basis as follows:

Encryption: Encrypt w/K1, Decrypt w/K2, Encrypt w/K3
Decryption: Decrypt w/K3, Encrypt w/K2, Decrypt w/K1

Triple-DES roughly doubles the effective key strength of DES. For further discussions on Triple-DES, see *Applied Cryptography: Protocols, Algorithms, and Source Code in C* by Bruce Schneier.

The `encr3des` module uses cipher-block chaining ("CBC"), as per RFC 2451 and has the following properties:

- **Key Size**: 192 bits. The single 192-bit key consists of three DES keys concatenated together in the _encryption_ (outbound) order. See `encrdes(7M)`. The `encr3des` module supports weak-key checking and parity-fixing to aid `pf_key(7P)`.
- **Block Size**: 64 bit.

**Export Restriction**

Triple DES has an effective key strength of approximately 112 bits and is only available inside the United States. Triple DES cannot be realistically weakened for use outside the United States.

**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>SUNWcryr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcryrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `ipseckey(1M), attributes(5), encrdes(7M), ipsec(7P), ipsecesp(7P), pf_key(7P)`
encrdes(7M)

NAME encrdes – DES-CBC Encryption Algorithm Module for IPsec

SYNOPSIS strmod/encrdes

DESCRIPTION This module implements the United States Data Encryption Standard (“DES”) for IPsec. encrdes uses cipher-block chaining (“CBC”), as per RFC 2405 and has the following properties:

Key Size 64 bits. 56 bit key, plus 8 parity bits. 7 bits of key are followed by one bit of odd parity. For example, the 56-bit key FF FF FF FF FF FF FF FF would be encoded as FE FE FE FE FE FE FE FE.

encrdes supports weak-key checking and parity-fixing to aid pf_key(7P).

Block Size 64 bits.

It is used by ESP.

Export Restriction DES with an actual key strength of 56 bits is only available inside the United States. DES has an effective key strength of approximately 56 bits and is only available inside the United States. DES cannot be realistically weakened for use outside the United States.

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>SUNWcryr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcryrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO ipseckey(1M), attributes(5), ipsec(7P), ipsecesp(7P), pf_key(7P)


eri(7D)

NAME
eri – eri Fast-Ethernet device driver

SYNOPSIS
/dev/eri

DESCRIPTION
The eri Fast Ethernet driver is a multi-threaded, loadable, clonable, STREAMS-based hardware driver supporting the connectionless Data Link Provider Interface dlpi(7P) over an eri Fast-Ethernet controller. Multiple eri devices installed within the system are supported by the driver.

The eri driver provides basic support for the eri hardware and handles the eri device. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The eri device provides 100Base-TX networking interfaces using the SUN RIO ASIC and an internal transceiver. The RIO ASIC provides the PCI interface and MAC functions. The physical layer functions are provided by the internal transceiver which connects to a RJ-45 connector.

The 100Base-TX standard specifies an auto-negotiation protocol to automatically select the mode and speed of operation. The internal transceiver is capable of performing auto-negotiation using the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. It also supports a forced-mode of operation under which the driver selects the mode of operation.

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

The eri driver is a “style 2” Data Link Service provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in <sys/dlpi.h>. Refer to dlpi(7P) for more information.

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

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are as follows:
- The maximum SDU is 1500 (ETHERMTU - defined in <sys/ethernet.h> ).
- The minimum SDU is 0.
- The dlsap address length is 8.
- The MAC type is DL_ETHER.
- The sap length values is -2, meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
The service mode is `DL_CLDLS`.
- Optional quality of service (QOS) is not currently supported so QOS fields are 0.
- The provider style is `DL_STYLE`.
- The version is `DL_VERSION_2`.
- The broadcast address value is Ethernet/IEEE broadcast address (0xffffff).

Once in the `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` to associate a particular SAP (Service Access Pointer) with the stream. The eri driver interprets the `sap` field within the `DL_BIND_REQ` as an Ethernet "type," therefore valid values for the `sap` field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If the user selects a `sap` with a value of 0, the receiver will be in IEEE 802.3 mode. All frames received from the media having an Ethernet type field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open Streams which are bound to `sap` value 0. If more than one Stream is in 802.3 mode, the frame will be duplicated and routed up multiple Streams as `DL_UNITDATA_IND` messages.

In transmission, the driver checks the `sap` field of the `DL_BIND_REQ` to determine if the value is 0 or if the Ethernet type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial `M_PROTO` mblk (message block), of all subsequent `DL_UNITDATA_REQ` messages, and transmits 802.3 frames that have this value in the MAC frame header length field.

The eri driver's DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte `sap` (type) component, producing an 8 byte DLSAP address. Applications should not hardcode to this particular implementation-specific DLSAP address format but use information returned in the `DL_INFO_ACK` primitive to compose and decompose DLSAP addresses. The `sap` length, full DLSAP length, and `sap`/physical ordering are included within the `DL_INFO_ACK`. The physical address length can be computed by subtracting the `sap` length from the full DLSAP address length or by issuing the `DL_PHYS_ADDR_REQ` to obtain the current physical address associated with the stream.

Once in the `DL_BOUND` state, the user may transmit frames on the Ethernet by sending `DL_UNITDATA_REQ` messages to the eri driver. The eri driver will route received Ethernet frames up all open and bound streams having a `sap` which matches the Ethernet type as `DL_UNITDATA_IND` messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the `DL_UNITDATA_REQ` and `DL_UNITDATA_IND` messages consists of both the `sap` (type) and physical (Ethernet) components.

---

**eri Primitives**

In addition to the mandatory connectionless DLPI message set, the driver also supports the following primitives:
The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS flag set in the dl_level field enables/disables reception of all promiscuous mode frames on the media, including frames generated by the local host. When used with the DL_PROMISC_SAP flag set, this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set, this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser, or EPERM is returned in the DL_ERROR_ACK. This primitive is destructive because it affects all current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, 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 eri driver performs auto-negotiation to select the mode and speed of the link, which can be in one of the following modes, as described in the 100Base-TX standard:

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

The auto-negotiation protocol automatically selects:

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

The auto-negotiation protocol does the following:

- Gets all modes of operation supported by the link partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities
The internal transceiver is capable of all of the operating speeds and modes listed above. By default, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the link partner.

For users who want to select the speed and mode of the link, the eri device supports programmable IPG (Inter-Packet Gap) parameters ipg1 and ipg2. Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mpbs and accordingly, IPG will be set to 9.6 or 0.96 microseconds.

The eri driver provides for setting and getting various parameters for the eri device. The parameter list includes current transceiver status, current link status, inter-packet gap, local transceiver capabilities and link partner capabilities.

The local transceiver has two set of capabilities: one set reflects hardware capabilities, which are read-only (RO) parameters. The second set reflects the values chosen by the user and is used in speed selection and possess read/write (RW) capability. At boot time, these two sets of capabilities will be the same. Because the current default value of these parameters can only be read and not modified, the link partner capabilities are also read only.

FILES
/dev/eri eri special character device.
/kernel/drv/sparcv9/eri.conf System wide default device driver properties
/kernel/drv/sparcv9/eri 64 bit device driver

SEE ALSO
ndd(1M), netstat(1M), driver.conf(4), hme(7D), qfe(7D), dlpi(7P)
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.

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.

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
tsxi-options=0x178 scsi-initiator-id=6;

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The above would set scsi-options for target 1 to 0x58 and for all other targets on this SCSI bus to 0x178. The physical pathname of the parent can be determined using the /devices tree or following the link of the logical device name:

    example# ls -l /dev/rdsk/c0t3d0s0
    lrwxrwxrwx 1 root root 88 Aug 22 13:29 /dev/rdsk/c0t3d0s0 -> 
    ../../devices/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/
    esp@f,800000/sd@3,0:a,raw

The register property values can be determined from prtconf(1M) output (-v option):

    esp, instance #0
    ....
    Register Specifications:
    Bus Type=0xf, Address=0x800000, Size=40
EXAMPLE 1 A sample of esp configuration file.  

(Continued)

To set scsi-options more specifically per target:

```plaintext
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 0x3f8 except for one specific disk type which will have scsi-options set to 0x58.

scsi-options specified per target ID has the highest precedence, followed by scsi-options per device type. To get the inquiry string run probe-scsi or probe-scsi-all command at the ok prompt before booting the system.

Global, for example, for all esp instances, scsi-options per bus has the lowest precedence.

The system needs to be rebooted before the specified scsi-options take effect.

FILES

/kernel/drv/esp     ELF Kernel Module
/kernel/drv/esp.conf Configuration file

ATTRIBUTES

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

<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 options</td>
</tr>
</tbody>
</table>

SEE ALSO

prtconf(1M), driver.conf(4), attributes(5), fas(7D), scsi_abort(9F),
scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_reset(9F),
scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S),
scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

OpenBoot Command Reference

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

ESP Technical Manuals, QLogic Corp.
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.

<name> message <n> from Target <m> garbled
  Target <m> sent message <name> of value <n> which the driver did not understand.

Target <n> rejects our message <name>
  Target <n> rejected a message sent by the driver.

Rejecting message <name> from Target <n>
  The driver rejected a message received from target <n>

Cmd dma error
  The driver was unable to send out command bytes.

Target <n> refused message resend
  The target did not accept a message resend.

Two-byte message <name> <value> rejected
  The driver does not accept this two-byte message.

Unexpected selection attempt
  An attempt was made to select this host adapter by another initiator.

Polled cmd failed (target busy)
  A polled command failed because the target did not complete outstanding commands within a reasonable time.

Polled cmd failed
  A polled command failed because of timeouts or bus errors.

Disconnected command timeout for Target <id>.<lun>
  A timeout occurred while target/lun was disconnected. This is usually a target f/w problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.

Disconnected tagged cmds (<n>) timeout for Target <id>.<lun>
  A timeout occurred while target/lun was disconnected. This is usually a target f/w problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.
Connected command timeout for Target <id>.<lun>
This is usually a SCSI bus problem. Check cables and termination.

Target <id>.<lun> reverting to async. mode
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id>.<lun> reducing sync. transfer rate
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Reverting to slow SCSI cable mode
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Reset SCSI bus failed
An attempt to reset the SCSI bus failed.

External SCSI bus reset
Another initiator reset the SCSI bus.

WARNINGS
The esp hardware does not support Wide SCSI mode. Only FAS-type esp’s support fast SCSI (10 MB/sec).

NOTES
The esp driver exports properties indicating per target the negotiated transfer speed (target<n>-sync-speed) and whether tagged queuing has been enabled (target<n>-TQ). The sync-speed property value is the data transfer rate in KB/sec. The target-TQ property has no value. The existence of the property indicates that tagged queuing has been enabled. Refer to prtconf(1M) (verbose option) for viewing the esp properties.

    dma, instance #3
    Register Specifications:
        Bus Type=0x2, Address=0x81000, Size=10
    esp, instance #3
    Driver software properties:
        name <target3-TQ> length <0> - <no value>.
        name <target3-sync-speed> length <4> value <0x00002710>.
        name <scsi-options> length <4> value <0x000003f8>.
        name <scsi-watchdog-tick> length <4> value <0x0000000a>.
        name <scsi-tag-age-limit> length <4> value <0x00000008>.
        name <scsi-reset-delay> length <4> value <0x000000bb8>.

154  man pages section 7: Device and Network Interfaces • Last Revised 7 Feb 1997
### NAME
fas - FAS SCSI Host Bus Adapter Driver

### SYNOPSIS
fas@bus-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.

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`
- `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.

### Driver Configuration

The target driver needs to set capabilities in the fas driver in order to enable some driver features. The target driver can query and modify these capabilities:
- `synchronous`
- `tagged-qing`
- `wide-xfer`
- `auto-rqsense`
- `qfull-retries`
- `qfull-retry-interval`

All other capabilities can only be queried.

By default, `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled, while `disconnect`, `synchronous`, and `untagged-qing` are enabled. These capabilities can only have binary values (0 or 1). The default value for `qfull-retries` is 10 and the default value for `qfull-retry-interval` is 100. The `qfull-retries` capability is a `uchar_t` (0 to 255) while `qfull-retry-interval` is a `ushort_t` (0 to 65535).

The target driver needs to enable `tagged-qing` and `wide-xfer` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified, because fas can queue commands even when `tagged-qing` is disabled.
Whenever there is a conflict between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

**EXAMPLE 1**

Create a file called `/kernel/drv/fas.conf` and add this line:

```bash
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):

```bash
name=“fas” parent=“/iommu@f,e0000000/sbus@f,e0001000”
  reg=3,0x8800000,0x10,3,0x8810000,0x40
  target1-scsi-options=0x58
  scsi-options=0x178 scsi-initiator-id=6;
```

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The example above sets `scsi-options` for target 1 to `0x58` and all other targets on this SCSI bus to `0x178`.

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/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):

```bash
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.

```bash
device-type-scsi-options-list=
  "TOSHIBA XMS701TASUN12XCD", "cd-scsi-options";
  cd-scsi-options = 0x0;
```
EXAMPLE 1 A sample fas.conf configuration file (Continued)

The above entry in /kernel/drv/fas.conf sets the scsi-options for all devices with inquiry string TOSHIBA XM5701TASUN12XCD to cd-scsi-options. To get the inquiry string, run the probe-scsi or probe-scsi-all command at the ok prompt before booting the system.

To set scsi-options more specifically per target:

target1-scsi-options=0x78;
device-type-scsi-options-list =
   "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;

The above sets scsi-options for target 1 to 0x78 and for all other targets on this SCSI bus to 0x3f8 except for one specific disk type which will have scsi-options set to 0x58.

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

The system needs to be rebooted before the specified scsi-options take effect.

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 3.x Command Reference Manual

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

QLogic Corporation, FAS366 Technical Manuals.
The messages described below are some that may appear on the system console, as well as being logged.

<table>
<thead>
<tr>
<th>fas(7D)</th>
<th><strong>DIAGNOSTICS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>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 &quot;fas%d&quot;, where &quot;%d&quot; is the instance number of the fas controller.</td>
<td></td>
</tr>
<tr>
<td>Device in slave-only slot</td>
<td></td>
</tr>
<tr>
<td>The SBus device has been placed in a slave-only slot and will not be accessible; move to non-slave-only SBus slot.</td>
<td></td>
</tr>
<tr>
<td>Device is using a hilevel intr</td>
<td></td>
</tr>
<tr>
<td>The device was configured with an interrupt level that cannot be used with this fas driver. Check the SBus device.</td>
<td></td>
</tr>
<tr>
<td>Cannot allocate soft state</td>
<td></td>
</tr>
<tr>
<td>Cannot alloc dma handle</td>
<td></td>
</tr>
<tr>
<td>Cannot alloc cmd area</td>
<td></td>
</tr>
<tr>
<td>Cannot create kmem_cache</td>
<td></td>
</tr>
<tr>
<td>Driver was unable to allocate memory for internal data structures.</td>
<td></td>
</tr>
<tr>
<td>Unable to map FAS366 registers</td>
<td></td>
</tr>
<tr>
<td>Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.</td>
<td></td>
</tr>
<tr>
<td>Cannot add intr</td>
<td></td>
</tr>
<tr>
<td>Driver could not add its interrupt service routine to the kernel.</td>
<td></td>
</tr>
<tr>
<td>Cannot map dma</td>
<td></td>
</tr>
<tr>
<td>Driver was unable to locate a DMA controller. This is an auto-configuration error.</td>
<td></td>
</tr>
<tr>
<td>Cannot bind cmdarea</td>
<td></td>
</tr>
<tr>
<td>Driver was unable to bind the DMA handle to an address.</td>
<td></td>
</tr>
<tr>
<td>Cannot create devctl minor node</td>
<td></td>
</tr>
<tr>
<td>Driver is unable to create a minor node for the controller.</td>
<td></td>
</tr>
<tr>
<td>Cannot attach</td>
<td></td>
</tr>
<tr>
<td>The driver was unable to attach; usually follows another warning that indicates why attach failed.</td>
<td></td>
</tr>
<tr>
<td>Disabled TQ since disconnects are disabled</td>
<td></td>
</tr>
<tr>
<td>Tagged queuing was disabled because disconnects were disabled in scsi-options.</td>
<td></td>
</tr>
<tr>
<td>Bad clock frequency</td>
<td></td>
</tr>
<tr>
<td>Check for bad hardware.</td>
<td></td>
</tr>
<tr>
<td>Sync of pkt (&lt;address&gt;) failed</td>
<td></td>
</tr>
<tr>
<td>Syncing a SCSI packet failed. Refer to scsi_sync_pkt(9F).</td>
<td></td>
</tr>
</tbody>
</table>
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 phase manage
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> 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> 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.
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 <0x000002710>.
    name <target3-scsi-options> length <4>
        value <0x000003f8>.
    name <target0-TQ> length <4>
        value <0x00000001>.
    name <pm_norm_pwr> length <4>
        value <0x00000001>.
    name <pm_timestamp> length <4>
        value <0x30040346>.
    name <scsi-options> length <4>
        value <0x000003f8>.
    name <scsi-watchdog-tick> length <4>
        value <0x0000000a>.
    name <scsi-tag-age-limit> length <4>
        value <0x00000002>.
    name <scsi-reset-delay> length <4>
        value <0x000000bb>.
Register Specifications:
    Bus Type=0x3, Address=0x880000, 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 `FBIOSVIDEO` control. `FBIOSVIDEO` and `FBIOGVIDEO` return the current video state.

The `FBIOSATTR` and `FBIOGATTR` `ioctl()` requests allow access to special features of newer frame buffers. They use the `fbattr` and `fbgaattr` structures.

Some color frame buffers support the `FBIOPUTCMAP` and `FBIOGETCMAP` `ioctl()` requests, which provide access to the colormap. They use the `fbcmap` structure.

Also, some framebuffers with multiple colormaps will either encode the colormap identifier in the high-order bits of the "index" field in the fbcmap structure, or use the `FBIOPUTCMAPI` and `FBIOGETCMAPI` `ioctl()` requests.

`FBIOVERTICAL` is used to wait for the start of the next vertical retrace period.

`FBIOVRTOFFSET` Returns the offset to a read-only vertical retrace page for those framebuffers that support it. This vertical retrace page may be mapped into user space with `mmap(2)`. The first word of the vertical retrace page (type unsigned int) is a counter that is incremented every time there is a vertical retrace. The user process can use this counter in a variety of ways.

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

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

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

### SEE ALSO
`ioctl(2), mmap(2), bwtwo(7D), cgeight(7D), cgfour(7D), cgsix(7D), cgthree(7D), cgtwo(7D)`
BUGS  The FBOSATTR 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 FBIOGPOS ioctl was incorrectly defined in previous operating systems, and older code running in binary compatibility mode may get incorrect results.
The `fcip` driver is a Fibre Channel upper layer protocol module for encapsulating IP (IPv4) and ARP datagrams over Fibre Channel. The `fcip` driver is a loadable, clonable, STREAMS driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)` over any Sun Fibre Channel transport layer-compliant host adapter.

The `fcip` driver complies with the RFC 2625 specification for encapsulating IP/ARP datagrams over Fibre Channel, and allows encapsulation of IPv4 only, as specified in RFC 2625. The `fcip` driver interfaces with the `fp(7D)` Sun Fibre Channel port driver.

The cloning character-special device `/dev/fcip` is used to access all Fibre Channel ports capable of supporting IP/ARP traffic on the system.

The `fcip` driver is a "style 2" Data Link Service Provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information on DLPI primitives.

An explicit `DL_ATTACH_REQ` message must be sent to associate the opened stream with a particular Fibre Channel port (ppa). The ppa ID is interpreted as an unsigned long data type and indicates the corresponding Fibre Channel port driver instance number. An error (`DL_ERROR_ACK`) is returned by the driver if the ppa field value does not correspond to a valid port driver instance number or if the Fibre Channel port is not `ONLINE`. Refer to `fp(7D)` for more details on the Fibre Channel port driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to a `DL_INFO_REQ` from the user are as follows:

- Maximum SDU is 65280 (defined in RFC 2625).
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is `DL_ETHER`.
- SAP length is `-2`.
- Service mode is `DL_CLDLS`.
- Optional quality of service (QOS) fields are set to 0.
- Provider style is `DL_STYLE2`.
- Provider version is `DL_VERSION_2`.
- Broadcast address value is `0xFFFFFFFF`.

Once in `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` to associate a particular SAP (Service Access Point) with the stream. The `fcip` driver DLSAP address format consists of the 6-byte physical address component followed immediately by the 2-byte SAP component producing an 8-byte DLSAP address. Applications should not be programmed to use this implementation-specific DLSAP address format, but use information returned in the `DL_INFO_ACK` primitive to...
compose and decompose DLSAP addresses. The SAP length, full DLSAP length, and SAP/physical ordering are included within the \texttt{DL\_INFO\_ACK}. The physical address length is the full DLSAP address length minus the SAP length. The physical address length can also be computed by issuing the \texttt{DL\_PHYS\_ADDR\_REQ} primitive to obtain the current physical address associated with the stream.

Once in the \texttt{DL\_BOUND} state, the user can transmit frames on the fibre by sending \texttt{DL\_UNITDATA\_REQ} messages to the \texttt{fcip} driver. The \texttt{fcip} driver will route received frames up any of the open and bound streams having a SAP which matches the received frame’s SAP type as \texttt{DL\_UNITDATA\_IND} messages. Received Fibre Channel frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the \texttt{DL\_UNITDATA\_REQ} and \texttt{DL\_UNITDATA\_IND} messages consists of both the SAP (type) and physical address (WorldWideName) components.

In Fibre Channel, \textit{multicasting} is defined as an optional service for Fibre Channel classes three and six only. If required, the Fibre Channel broadcast service can be used for multicasting. The \texttt{RFC 2625} specification does not support IP multicasting or promiscuous mode.

The \texttt{fcip} driver will use the FARP Fibre Channel Extended Link Service (ELS), where supported, to resolve WorldWide Names (MAC address) to FC Port Identifiers(Port ID). The \texttt{fcip} driver also supports InARP to resolve WorldWide Name and Port_ID to an IP address.

**FILES**

\begin{itemize}
  \item [/dev/fcip] fcip character-special device
  \item [/kernel/drv/fcip] 32-bit ELF kernel driver
  \item [/kernel/drv/sparcv9/fcip] 64-bit ELF kernel driver
  \item [/kernel/drv/fcip.conf] fcip driver configuration file
\end{itemize}

**ATTRIBUTES**

See \texttt{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>
<tr>
<td>Availability</td>
<td>SUNWfcip</td>
</tr>
</tbody>
</table>

**SEE ALSO**

\begin{itemize}
  \item netstat(1M), prtconf(1M), driver.conf(4), fp(7D), dlpi(7P)
  \item \textit{Writing Device Drivers}
  \item \textit{IP and ARP over Fibre Channel, RFC 2625} M. Rajagopal, R. Bhagwat, W. Rickard.
  \item Gadzoox Networks, June 1999
\end{itemize}
ANSI X3.230-1994, Fibre Channel Physical and Signalling Interface (FC-PH)

ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)

NOTES

If you use a Fibre Channel adapter with two or more ports that each share a common Node WorldWideName, the fcip driver will likely attach to the first port on the adapter.

RFC 2625 requires that both source and destination WorldWideNames have their 4 bit NAA identifiers set to binary ‘0001,’ indicating that an IEEE 48–bit MAC address is contained in the lower 48 bits of the network address fields. For additional details, see the RFC 2625 specification.
fcp(7D)

NAME  fcp – Fibre Channel protocol driver

DESCRIPTION  The fcp driver is the upper layer protocol that supports mechanisms for transporting SCSI-3 commands over Fibre Channel. The fcp driver, which interfaces with the Sun Fibre Channel transport library fctl(7D), supports the standard functions provided by the SCSA interface.

FILES  
/kernel/drv/fcp
    32-bit ELF kernel driver
/kernel/drv/sparcv9/fcp
    64-bit ELF kernel 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>SPARC</td>
</tr>
<tr>
<td>Interface stability</td>
<td>Unknown</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWfcp</td>
</tr>
</tbody>
</table>

SEE ALSO  prtconf(1M), driver.conf(4), fctl(7D), fcip(7D), fp(7D), usoc(7D)

Writing Device Drivers

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994
Fibre Channel Generic Services (FC-GS-2) Project 1134-D
Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996
Fibre Channel Protocol for SCSI (FCP) ANSI X3.269-1996
SCSI-3 Architecture Model (SAM) Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) ANSI X3.270-1996
Fabric Loop Attachment (FC-FLA), NCITS TR-20:1998
NAME  
fctl – Sun Fibre Channel transport library

DESCRIPTION  
The fctl kernel module interfaces the Sun Fibre Channel upper layer protocol (ULP) mapping modules with Sun Fibre Channel adapter (FCA) drivers. There are no user-configurable options for this module.

FILES  
/kernel/misc/fctl
32-bit ELF kernel module
/kernel/misc/sparcv9/fctl
64-bit 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>
<tr>
<td>Interface stability</td>
<td>Unknown</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWfctl</td>
</tr>
</tbody>
</table>

SEE ALSO  
attributes(5), fp(7D)
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 IA based systems.

The default partitions for the floppy driver are:

a) All cylinders except the last
b) Only the last cylinder
c) Entire diskette

The `fd` driver autosenses the density of the diskette.

When the floppy is first opened the driver looks for a SunOS label in logical block 0 of the diskette. If attempts to read the SunOS label fail, the open will fail. If block 0 is read successfully but a SunOS label is not found, auto-sensed geometry and default partitioning are assumed.

The `fd` driver supports both block and "raw" interfaces.

The block files (`/dev/diskette*`) access the diskette using the system’s normal buffering mechanism and may be read and written without regard to physical diskette records.

There is also a "raw" (`/dev/rdiskette*`) interface that provides for direct transmission between the diskette and the user’s read or write buffer. A single `read(2)` or `write(2)` call usually results in one I/O operation; therefore raw I/O is considerably more efficient when larger blocking factors are used. A blocking factor of no less than 8 Kbytes is recommended. See the Notes section, below, for information on the number of sectors per track.

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

<table>
<thead>
<tr>
<th>SPARC</th>
<th>1.7 Mbyte density</th>
<th>high density</th>
<th>double density</th>
<th>medium density</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80 cylinders, 21 sectors per track, 1.7 Mbyte capacity</td>
<td>80 cylinders, 18 sectors per track, 1.44 Mbyte capacity</td>
<td>80 cylinders, 9 sectors per track, 720 Kbyte capacity</td>
<td>77 cylinders, 8 sectors per track, 1.2 Mbyte capacity (sun4m only)</td>
</tr>
</tbody>
</table>
For 5.25" double-sided diskettes on IA platforms, the densities listed below are supported:

**5.25" Diskettes**

1. extended density 80 cylinders, 36 sectors per track, 2.88 Mbyte capacity
2. 1.7 Mbyte density 80 cylinders, 21 sectors per track, 1.7 Mbyte capacity
3. high density 80 cylinders, 18 sectors per track, 1.44 Mbyte capacity
4. double density 80 cylinders, 9 sectors per track, 760 Kbyte capacity

**SPARC**

5.25" diskettes are not supported on SPARC platforms.

**IA**

1. high density 80 cylinders, 15 sectors per track, 1.2 Mbyte capacity
2. double density 40 cylinders, 9 sectors per track, 360 Kbyte capacity
3. double density 40 cylinders, 8 sectors per track, 320 Kbyte capacity
4. quad density 80 cylinders, 9 sectors per track, 720 Kbyte capacity
5. double density 40 cylinders, 16 sectors per track (256 bytes per sector), 320 Kbyte capacity
6. double density 40 cylinders, 4 sectors per track (1024 bytes per sector), 320 Kbyte capacity

**ERRORS**

1. **EBUSY**
   
   During opening, the partition has been opened for exclusive access and another process wants to open the partition. Once open, this error is returned if the floppy disk driver attempted to pass a command to the floppy disk controller when the controller was busy handling another command. In this case, the application should try the operation again.

2. **EFAULT**
   
   An invalid address was specified in an ioctl command (see f dio(7I)).

3. **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 f dio(7I)).

4. **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.

5. **ENOSPC**
   
   An attempt was made to write past the end of the diskette.

6. **ENOTTY**
   
   The floppy disk driver does not support the requested ioctl functions (see f dio(7I)).
### fd(7D)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENXIO</td>
<td>The floppy disk device does not exist or the device is not ready.</td>
</tr>
<tr>
<td>EROFS</td>
<td>The floppy disk device is opened for write access and the diskette in the drive is write protected.</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>The floppy disk device does not support the requested ioctl function (FDEJECT).</td>
</tr>
</tbody>
</table>

**IA Only**

The driver attempts to initialize itself using the information found in the configuration file, `/platform/i86pc/kernel/drv/fd.conf`.

```plaintext
name="fd" parent="fdc" unit=0;
name="fd" parent="fdc" unit=1;
```

**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
- `/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`

**IA**

- `/platform/i86pc/kernel/drv/fd` driver module
- `/platform/i86pc/kernel/drv/fd.conf` configuration file for floppy driver
floppy-controller driver module
/configuration file for the floppy-controller
/structs and definitions for IA floppy devices
/structs and definitions for IA floppy media

IA First Drive
/dev/diskette
/dev/diskette0
/dev/rdiskette
/dev/rdiskette0

device file
device file
raw device file
raw device file

For ucb Compatibility
/dev/fd0[a-c]
/dev/rfd0[a-c]
/vol/dev/diskette0
/vol/dev/rdiskette0
/vol/dev/aliases/floppy0

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

IA Second Drive
/dev/diskette1
/dev/rdiskette1

device file
raw device file

For ucb Compatibility
/dev/fd1[a-c]
/dev/rfd1[a-c]
/vol/dev/diskette1
/vol/dev/rdiskette1
/vol/dev/aliases/floppy1

directory containing volume management
character device file
directory containing the volume
management raw character device file
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)

All Platforms

fd<n>: <command name> failed (<sr1> <sr2> <sr3>)
The <command name> failed after several retries on drive <n>. The three hex
values in parenthesis are the contents of status register 0, status register 1, and
status register 2 of the Intel 8272, the Intel 82072, and the Intel 82077 Floppy Disk
Controller on completion of the command, as documented in the data sheet for that
part. This error message is usually followed by one of the following, interpreting
the bits of the status register:

fd<n>: not writable
fd<n>: crc error blk <block number>
There was a data error on <block number>.
fd<n>: bad format
fd<n>: timeout
fd<n>: drive not ready
fd<n>: unformatted diskette or no diskette in drive
fd<n>: block <block number> is past the end!
(nblk=<total number of blocks>)
The operation tried to access a block number that is greater than
the total number of blocks.
fd<n>: b_bcount 0x<op_size> not % 0x<sect_size>
The size of an operation is not a multiple of the sector size.
fd<n>: overrun/underrun
fd<n>: host bus error. There was a hardware error on a system
bus.

SPARC Only

Overrun/underrun errors occur when accessing a diskette while the system is heavily
loaded. Decrease the load on the system and retry the diskette access.

NOTES

3.5" high density diskettes have 18 sectors per track and 5.25" high density diskettes
have 15 sectors per track. They can cross a track (though not a cylinder) boundary
without losing data, so when using dd(1M) or read(2)/write(2) calls to or from the
“raw” diskette, you should specify bs=18k or multiples thereof for 3.5" diskettes, and
bs=15k or multiples thereof for 5.25" diskettes.

The SPARC fd driver is not an unloadable module.

Under Solaris (Intel Platform Edition), the configuration of the floppy drives is
specified in CMOS configuration memory. Use the BIOS setup program or an EISA

174  man pages section 7: Device and Network Interfaces  •  Last Revised 24 Feb 1998
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.

FDDEFGECH | IA based systems: This ioctl() forces the floppy driver to restore the diskette and drive characteristics and geometry, and partition information to default values based on the device configuration.

FDGETCHANGE | The argument is a pointer to an int. This ioctl() returns the status of the diskette-changed signal from the floppy interface. The following defines are provided for cohesion.

Note: For IA based systems, use FDGC_DETECTED (which is available only on IA based systems) instead of FDGC_HISTORY.

/ *
     * Used by FDGETCHANGE, returned state of the sense disk change bit.
     */
#define FDGC_HISTORY 0x01 /* disk has changed since 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:

field value
fdc_medium  0
fdc_transfer_rate  500
fdc_ncyl  80
fdc_nhead  2
fdc_sec_size  512
fdc_secptrack  18
fdc_steps  1  { This field doesn't apply. }

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

FDIOCMD The argument is a pointer to an fd_cmd structure (described below). This ioctl() allows access to the floppy diskette using the floppy device driver. Only the FDCMD_WRITE, FDCMD_READ, and FDCMD_FORMAT_TR commands are currently available.

struct fd_cmd {
    ushort_t fdc_cmd; /* command to be executed */
    int fdc_flags; /* execution flags (IA only) */
};
Please note that when using FDRAW_SEEK or FDRAW_REZERO, the driver automatically issues a FDRAW_SENSE_INT command to clear the interrupt from the FDRAW_SEEK or the FDRAW_REZERO. The result bytes returned by these commands are the results from the FDRAW_SENSE_INT command. Please see the floppy-controller data sheet for more details on FDRAW_SENSE_INT.

/*
 * Used by FDRAW
 */
struct fd_raw {
char fdr_cmd[10]; /* user-supplied command bytes */
short fdr_cnum; /* number of command bytes */
char fdr_result[10]; /* controller-supplied result bytes */
}
FDIO command structures

```c
struct fdio_argfd {
    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)`:  

```plaintext
FBIOPUTCMAP, FBIOGETCMAP, FBIOSVIDEO, FBIOGVIDEO, FBIOVERTICAL,  
FBIOSCURSOR, FBIOGSCURSOR, 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

<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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO  driver.conf(4), sysbus(4), attributes(5)
fp(7d)

NAME  fp – Sun Fibre Channel port driver

DESCRIPTION  The fp driver is a Sun Fibre Channel nexus driver that enables Fibre Channel topology discovery, device discovery, Fibre Channel adapter port management and other capabilities through well-defined Fibre Channel adapter driver interfaces.

The fp driver requires the presence of a fabric name server in fabric and public loop topologies to discover fibre channel devices. In private loop topologies, the driver discovers devices by performing PLOGI to all valid AL_PAs, provided that devices do not participate in LIRP and LILP stages of loop initialization.

FILES  /kernel/drv/fp
       32-bit ELF kernel driver

       /kernel/drv/sparcv9/fp
       64-bit ELF kernel driver

       /kernel/drv/fp.conf
       fp 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>
<tr>
<td>Interface stability</td>
<td>Unknown</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWfctl</td>
</tr>
</tbody>
</table>

SEE ALSO  prtconf(1M), driver.conf(4), fctl(7D)

Writing Device Drivers,

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994,

Fibre Channel Generic Services (FC-GS-2) Project 1134-D,

Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996,

Fibre Channel Protocol for SCSI (FCP) ANSI X3.269-1996,

SCSI-3 Architecture Model (SAM) Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) ANSI X3.270-1996,

SCSI Direct Attach (FC-PLDA) NCITS TR-19:1998,

Fabric Loop Attachment (FC-FLA), NCITS TR-20:1998
The `ge` Gigabit-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)` over GEM, SBus and PCI Gigabit-Ethernet add-in adapters. Multiple GEM-based adapters installed within the system are supported by the driver. The `ge` driver provides basic support for the GEM-based Ethernet hardware and handles the `sunw,sbus-gem` (SBus GEM) and `pci108e,2bad` (PCI GEM) devices. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

The GEM device provides 1000BASE-SX networking interfaces using the GEM ASIC external SERDES and fiber optical transceiver. The GEM ASIC provides the appropriate bus interface, MAC functions and physical code sub-layer (PCS) functions. The external SERDES connects to a fiber transceiver and provides the physical connection.

The 1000Base-SX standard specifies an auto-negotiation protocol to automatically select the mode of operation. In addition to duplex operation, the GEM ASIC can auto-negotiate for `IEEE 802.3x` frame based flow control capabilities. The GEM PCS is capable of performing auto-negotiation using the remote (or link partner) link end and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on priorities. The `ge` driver also supports forced-mode operation under which the driver selects the mode of operation.

The cloning character-special device `/dev/ge` is used to access all `ge` controllers installed within the system.

The `ge` driver is a Style 2 data link service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information.

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

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

- Maximum SDU is 1500 (`ETHERMTU` defined in `<sys/ethernet.h>`).
- Minimum SDU is 0.
- dlsap address length is 8.
- MAC type is `DL_ETHER`.

Device and Network Interfaces 183
sap length value is \(-2\), meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.

- Service mode is \texttt{DL\_CLDLS}.
- Quality of service (QOS) is not supported; accordingly, QOS fields are 0.
- Provider style is \texttt{DL\_STYLE2}.
- Version is \texttt{DL\_VERSION\_2}.
- Broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFFFF).

Once in the \texttt{DL\_ATTACHED} state, you must send a \texttt{DL\_BIND\_REQ} to associate a particular Service Access Pointer (SAP) with the stream. The \texttt{ge} driver interprets the sap field within the \texttt{DL\_BIND\_REQ} as an Ethernet type; accordingly, valid values for the sap field are in the \([0-0xFFFF]\) range. Only one Ethernet type can be bound to the stream at any time.

If you select a sap with a value of 0, the receiver will be in 802.3 mode. All frames received from the media with a type field in the range \([0-1500]\) are assumed to be 802.3 frames and are routed up all open streams bound to sap value 0. If more than one stream is in 802.3 mode, the frame will be duplicated and routed up multiple streams as \texttt{DL\_UNITDATA\_IND} messages.

In transmission, the driver checks the sap field of the \texttt{DL\_BIND\_REQ} to determine if the sap value is 0 and the destination type field is in the range \([0-1500]\). If either is true, the driver computes the length of the message, not including initial \texttt{M\_PROTO} mblk (message block), of all subsequent \texttt{DL\_UNITDATA\_REQ} messages and transmits 802.3 frames of that value in the MAC frame header length field.

The \texttt{ge} driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should not hard code to this particular implementation-specific DLSAP address format, but use information returned in the \texttt{DL\_INFO\_ACK} primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length and sap physical ordering are included within the \texttt{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 \texttt{DL\_PHYS\_ADDR\_REQ} to obtain the current physical address associated with the stream.

Once in the \texttt{DL\_BOUND} state, you may transmit frames on the Ethernet by sending \texttt{DL\_UNITDATA\_REQ} messages to the \texttt{ge} driver. The \texttt{ge} driver will route received Ethernet frames up all open and bound streams having a sap which matches the Ethernet type as \texttt{DL\_UNITDATA\_IND} messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the \texttt{DL\_UNITDATA\_REQ} and \texttt{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 \texttt{ge} primitives.
The `DL_ENABMULTI_REQ` and `DL_DISABMULTI_REQ` primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The `DL_ENABMULTI_REQ` and `DL_DISABMULTI_REQ` primitives are accepted by the driver in any state following `DL_ATTACHED`.

With the `DL_PROMISC_PHYS` flag set in the `dl_level` field, the `DL_PROMISCON_REQ` and `DL_PROMISCOFF_REQ` primitives enable/disable reception of all promiscuous mode frames on the media including frames generated by the local host. When used with the `DL_PROMISC_SAP` flag set, this enables/disables reception of all `sap` (Ethernet type) values. When used with the `DL_PROMISC_MULTI` flag set, this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other `sap` and physical level configurations on the stream or other streams.

The `DL_PHYS_ADDR_REQ` primitive returns the six octet Ethernet address currently associated (attached) to the stream in the `DL_PHYS_ADDR_ACK` primitive. This primitive is valid only in states following a successful `DL_ATTACH_REQ`.

The `DL_SET_PHYS_ADDR_REQ` primitive changes the 6 octet Ethernet address currently associated (attached) to the stream. The credentials of the process which originally opened the stream must be superuser or `EPERM` is returned in the `DL_ERROR_ACK`. The `DL_SET_PHYS_ADDR_REQ` primitive is destructive and affects all other current and future streams attached to this device. A `M_ERROR` is sent up all other streams attached to the device when `DL_SET_PHYS_ADDR_REQ` is successful on the stream. Once changed, all streams subsequently opened and attached to the device will obtain the new physical address. Once changed, the physical address will remain until `DL_SET_PHYS_ADDR_REQ` is used to change the physical address again or the system is rebooted, whichever comes first.

### ge DRIVER
By default, the `ge` driver performs auto-negotiation to select the mode and flow control capabilities of the link. The link can be in one of the following modes:

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- Symmetric pause
- Asymmetric pause

Speeds and modes are described in the 1000Base-TX standard.

The auto-negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Flow control capability (Symmetric and/or Asymmetric)

The auto-negotiation protocol:

- Gets all the modes of operation supported by the link partner.
- Advertises its capabilities to the link partner.
Selects the highest common denominator mode of operation based on the priorities.

When by default, auto-negotiation is used to bring up the link and select the common mode of operation with the link partner, the GEM hardware is capable of all of the operating modes listed above. The PCS also supports forced-mode of operation under which the driver can select the mode of operation and flow control capabilities using the ndd utility.

The GEM device also supports programmable Inter-Packet Gap (IPG) parameters ipg1 and ipg2. By default, the driver sets ipg1 to 8 byte-times and ipg2 to 4 byte-times, (the standard values.) You may want to alter these values from the standard 1000 Mbps IPG set to 0.096 microseconds.

The PCS has two set of capabilities. One set reflects the capabilities of the hardware and are read-only. The second set are read/write and are used in speed selection and reflect the values you choose. At boot time, both sets will be the same. The link partner capabilities are read only and cannot be modified.

FILES
/dev/ge  ge special character device
/kernel/drv/ge.conf  System wide default device driver properties

SEE ALSO
ndd(1M), netstat(1M), driver.conf(4), dlpi(7P), le(7D), hme(7D), qfe(7D)
GLD is a multi-threaded, clonable, loadable kernel module providing support for Solaris Local Area Network device drivers.

Local Area Network (LAN) device drivers in Solaris are STREAMS-based drivers that use the Data Link Provider Interface (DLPI) to communicate with network protocol stacks. These protocol stacks use the network drivers to send and receive packets on a local area network. A network device driver, therefore, must implement and adhere to the requirements imposed by the DDI/DKI specification, the STREAMS specification, the DLPI interface specification, and the programmatic interface of the device itself.

GLD implements most of the STREAMS functions and DLPI functionality required of a Solaris 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 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 calls into the GLD entry points of the device-specific driver through pointers provided to GLD by the device-specific driver when it registered itself with GLD.

The GLD facility currently supports devices of type DL_ETHER, DL_TPR, and DL_FDDI. GLD drivers are expected to process fully-formed MAC-Layer packets, and should not perform any Logical Link Control (LLC) handling.

In some cases it may be necessary or desirable to implement a full DLPI-compliant driver without using the GLD facility. This will be the case, for example, for devices that are not IEEE 802-style LAN devices, or where a device type or DLPI service not supported by GLD is required.

For devices designated type DL_ETHER, GLD provides support for both Ethernet V2 and IEEE 802.3 / ISO 8802-3 packet processing. Ethernet V2 enables a data link service user to access and use any of a variety of conforming data link service providers without special knowledge of the provider’s protocol. A Service Access Point (SAP) is the point through which the user communicates with the service provider.

Streams bound to SAP values in the range [0-255] are treated as equivalent and denote that the user wishes to use 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 the
14-byte MAC header, of each subsequent DL_UNITDATA_REQ message on that stream, and transmits 802.3 frames having that length in the MAC frame header type field. Such lengths will never exceed 1500.

Furthermore, 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 that are in 802.3 mode, i.e. are bound to a SAP value in the [0-255] range. If more than one stream is in 802.3 mode, the incoming frame will be duplicated and routed up each such stream.

Streams bound to SAP values > 1500 receive incoming packets whose Ethernet MAC header type value exactly matches the value of the SAP to which the Stream is bound.

For media types DL_TPR and DL_FDDI GLD implements minimal SNAP processing for any stream bound to a SAP value greater than 255. SAP values in the range [0-255] are LLC SAP values, and are carried naturally by the media packet format. However, SAP values greater than 255 require a SNAP (Sub-Net Access Protocol) header, under the LLC header, to carry the 16-bit Ethernet V2-style SAP value.

SNAP headers are carried under LLC headers with destination SAP 0xAA. For outgoing packets with SAP values greater than 255, GLD creates an LLC+SNAP header that always looks like:

```
AA AA 03 00 00 00 XX XX
```

where "XX XX" represents the 16-bit SAP, corresponding to the Ethernet V2 style "type". This is the only class of SNAP header supported — non-zero OUI fields, and LLC control fields other than 03, are considered to be LLC packets with SAP 0xAA. Clients wishing to use SNAP formats other than this one must use LLC and bind to SAP 0xAA.

Incoming packets are examined to ascertain whether they fall into the format specified above. Packets that do will be matched to Streams bound to the packet’s 16-bit SNAP type, as well as being considered to match the LLC SNAP SAP 0xAA.

Packets received for any LLC SAP are passed up all Streams that are bound to an LLC SAP, just as described for media type DL_ETHER above.

For type DL_TPR devices, GLD implements minimal support for Source Routing. Source Routing is a mechanism by which a station sending a packet across a bridged medium specifies, in the packet MAC header, Routing Information that determines the route that the packet will take through the bridged network.

Functionally, the Source Routing support provided by GLD learns routes, solicits and responds to requests for information about possible multiple routes, and selects among multiple routes available to it. It adds Routing Information Fields to the MAC headers of outgoing packets, and recognizes such fields in incoming packets.

---

**Types DL_TPR and DL_FDDI: SNAP processing**

For media types DL_TPR and DL_FDDI GLD implements minimal SNAP processing for any stream bound to a SAP value greater than 255. SAP values in the range [0-255] are LLC SAP values, and are carried naturally by the media packet format. However, SAP values greater than 255 require a SNAP (Sub-Net Access Protocol) header, under the LLC header, to carry the 16-bit Ethernet V2-style SAP value.

SNAP headers are carried under LLC headers with destination SAP 0xAA. For outgoing packets with SAP values greater than 255, GLD creates an LLC+SNAP header that always looks like:

```
AA AA 03 00 00 00 XX XX
```

where "XX XX" represents the 16-bit SAP, corresponding to the Ethernet V2 style "type". This is the only class of SNAP header supported — non-zero OUI fields, and LLC control fields other than 03, are considered to be LLC packets with SAP 0xAA. Clients wishing to use SNAP formats other than this one must use LLC and bind to SAP 0xAA.

Incoming packets are examined to ascertain whether they fall into the format specified above. Packets that do will be matched to Streams bound to the packet’s 16-bit SNAP type, as well as being considered to match the LLC SNAP SAP 0xAA.

Packets received for any LLC SAP are passed up all Streams that are bound to an LLC SAP, just as described for media type DL_ETHER above.

For type DL_TPR devices, GLD implements minimal support for Source Routing. Source Routing is a mechanism by which a station sending a packet across a bridged medium specifies, in the packet MAC header, Routing Information that determines the route that the packet will take through the bridged network.

Functionally, the Source Routing support provided by GLD learns routes, solicits and responds to requests for information about possible multiple routes, and selects among multiple routes available to it. It adds Routing Information Fields to the MAC headers of outgoing packets, and recognizes such fields in incoming packets.
GLD’s Source Routing support does not implement the full Route Determination Entity (RDE) specified in ISO 8802-2 (IEEE 802.2) Section 9. However, it is designed to interoperate with any such implementations that may exist in the same (or a bridged) network.

GLD implements both Style 1 and Style 2 providers. A Physical Point of Attachment (PPA) is the point at which a system attaches itself to a physical communication medium. All communication on that physical medium funnels through the PPA. The Style 1 provider attaches the stream to a particular PPA based on the major/minor device that has been opened. The Style 2 provider requires the DLS user to explicitly identify the desired PPA using DL_ATTACH_REQ. In this case, open(9E) creates a stream between the user and GLD, and DL_ATTACH_REQ subsequently associates a particular PPA with that stream. Style 2 is denoted by a minor number of zero. If a device node is opened whose minor number is not zero, that denotes Style 1, and the associated PPA is the minor number minus 1. In both Style 1 and Style 2 opens, the device is cloned.

GLD implements the following DLPI primitives:

The DL_INFO_REQ primitive requests information about the DLPI stream. The message consists of one M_PROTO message block. GLD returns device-dependent values in the DL_INFO_ACK response to this request, based on information the GLD-based driver passed to gld_register(). However GLD returns the following values on behalf of all GLD-based drivers:

- The version is DL_VERSION_2.
- The service mode is DL_CLDLS — GLD implements connectionless-mode service.
- The provider style is DL_STYLE1 or DL_STYLE2, depending on how the stream was opened.
- 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. Upon completion, the state changes from DL_UNATTACHED to DL_UNBOUND. The message consists of one M_PROTO message block. This request may not be issued when using the driver in Style 1 mode; streams opened using Style 1 are already attached to a PPA by the time the open completes.

The DL_DETACH_REQ primitive requests to detach the PPA from the stream. This is only allowed if the stream was opened using Style 2.

The DL_BIND_REQ and DL_UNBIND_REQ primitives bind and unbind a DLSAP to the stream. The PPA associated with each stream will have been initialized upon completion of the processing of the DL_BIND_REQ. Multiple streams may be bound to the same SAP; each such stream receives a copy of any packets received for that SAP.
The `DL_ENABMULTI_REQ` and `DL_DISABMULTI_REQ` primitives enable and disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The stream must be attached to a PPA for these primitives to be accepted.

The `DL_PROMISCON_REQ` and `DL_PROMISCOFF_REQ` primitives enable and disable promiscuous mode on a per-stream basis, either at a physical level or at the SAP level. The DL Provider will route all received messages on the media to the DLS User until either a `DL_DETACH_REQ` or a `DL_PROMISCOFF_REQ` is received or the stream is closed. Physical level promiscuous mode may be specified for all packets on the medium, or for multicast packets only. The stream must be attached to a PPA for these primitives to be accepted.

The `DL_UNITDATA_REQ` primitive is used to send data in a connectionless transfer. Because this is an unacknowledged service, there is no guarantee of delivery. The message consists of one `M_PROTO` message block followed by one or more `M_DATA` blocks containing at least one byte of data.

The `DL_UNITDATA_IND` type is used when a packet is received and is to be passed upstream. The packet is put into an `M_PROTO` message with the primitive set to `DL_UNITDATA_IND`.

The `DL_PHYS_ADDR_REQ` primitive returns the MAC address, currently associated with the PPA attached to the stream, in the `DL_PHYS_ADDR_ACK` primitive. When using style 2, this primitive is only valid following a successful `DL_ATTACH_REQ`.

The `DL_SET_PHYS_ADDR_REQ` primitive changes the MAC address currently associated with the PPA attached to the stream. This primitive affects all other current and future streams attached to this device. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. The new physical address will remain in effect until this primitive is used to change the physical address again or the driver is reloaded. This primitive will only succeed when no Stream currently attached to the selected device is bound (i.e. using `DL_BIND_REQ`).

GLD implements the ioctl `ioc_cmd` function described below.

The `DLIOCRAW` ioctl function is used by some DLPI applications, most notably the `snoop` command. The `DLIOCRAW` command puts the stream into a raw mode, which, on receive, causes the full MAC-level packet to be sent upstream in an `M_DATA` message instead of it being transformed into the `DL_UNITDATA_IND` form normally used for reporting incoming packets. Packet SAP filtering is still performed on streams that are in raw mode; if a stream user wants to receive all incoming packets it must also select the appropriate promiscuous modes. After successfully selecting raw mode, the application is also allowed to send fully formatted packets to the driver as `M_DATA` messages for transmission. `DLIOCRAW` takes no arguments. Once enabled, the stream remains in this mode until closed.
Solaris network drivers must implement statistics variables. GLD itself tallies some network statistics, but other statistics must be counted by each GLD-based driver. GLD provides support for GLD-based drivers to report a standard set of network driver statistics. Statistics are reported by GLD using the `kstat(7D)` and `kstat(9S)` mechanism. All statistics are maintained as unsigned, and all are 32 bits unless otherwise noted.

GLD maintains and reports the following statistics.

- `rbytes64`: Total bytes successfully received on the interface (64 bits).
- `rbytes`: Total bytes successfully received on the interface.
- `obytes64`: Total bytes requested to be transmitted on the interface (64 bits).
- `obytes`: Total bytes requested to be transmitted on the interface.
- `ipackets64`: Total packets successfully received on the interface (64 bits).
- `ipackets`: Total packets successfully received on the interface.
- `opackets64`: Total packets requested to be transmitted on the interface (64 bits).
- `opackets`: Total packets requested to be transmitted on the interface.
- `multircv`: Multicast packets successfully received, including group and functional addresses (long).
- `multixmt`: Multicast packets requested to be transmitted, including group and functional addresses (long).
- `brdcstrcv`: Broadcast packets successfully received (long).
- `brdcstxmt`: Broadcast packets requested to be transmitted (long).
- `unknowns`: Valid received packets not accepted by any stream.
- `noxmbuf`: Packets discarded on output because transmit buffer was busy, or no buffer could be allocated for transmit.
- `blocked`: Times a received packet could not be put up a stream because the queue was flow controlled.
- `xmtretry`: Times transmit was retried after having been delayed due to lack of resources.
- `promisc`: Current “promiscuous” state of the interface.

The device dependent driver may count the following statistics.
### gld(7D)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ifspeed</code></td>
<td>Current estimated bandwidth of the interface in bits per second (64 bits).</td>
</tr>
<tr>
<td><code>media</code></td>
<td>Current media type in use by the device.</td>
</tr>
<tr>
<td><code>intr</code></td>
<td>Times interrupt handler was called and claimed the interrupt.</td>
</tr>
<tr>
<td><code>norcvbuf</code></td>
<td>Times a valid incoming packet was known to have been discarded because no buffer could be allocated for receive.</td>
</tr>
<tr>
<td><code>ierrors</code></td>
<td>Total packets received that couldn’t be processed because they contained errors.</td>
</tr>
<tr>
<td><code>oerrors</code></td>
<td>Total packets that weren’t successfully transmitted because of errors.</td>
</tr>
<tr>
<td><code>missed</code></td>
<td>Packets known to have been dropped by the hardware on receive.</td>
</tr>
<tr>
<td><code>uflo</code></td>
<td>Times FIFO underflowed on transmit.</td>
</tr>
<tr>
<td><code>oflo</code></td>
<td>Times receiver overflowed during receive.</td>
</tr>
</tbody>
</table>

The following group of statistics applies to networks of type `DL_ETHER`; these are maintained by device-specific drivers of that type, as above.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>align_errors</code></td>
<td>Packets received with framing errors (not an integral number of octets).</td>
</tr>
<tr>
<td><code>fcs_errors</code></td>
<td>Packets received with CRC errors.</td>
</tr>
<tr>
<td><code>duplex</code></td>
<td>Current duplex mode of the interface.</td>
</tr>
<tr>
<td><code>carrier_errors</code></td>
<td>Times carrier was lost or never detected on a transmission attempt.</td>
</tr>
<tr>
<td><code>collisions</code></td>
<td>Ethernet collisions during transmit.</td>
</tr>
<tr>
<td><code>ex_collisions</code></td>
<td>Frames where excess collisions occurred on transmit, causing transmit failure.</td>
</tr>
<tr>
<td><code>tx_late_collisions</code></td>
<td>Times a transmit collision occurred late (after 512 bit times).</td>
</tr>
<tr>
<td><code>defer_xmts</code></td>
<td>Packets without collisions where first transmit attempt was delayed because the medium was busy.</td>
</tr>
<tr>
<td><code>first_collisions</code></td>
<td>Packets successfully transmitted with exactly one collision.</td>
</tr>
<tr>
<td><code>multi_collisions</code></td>
<td>Packets successfully transmitted with multiple collisions.</td>
</tr>
<tr>
<td><code>sqe_errors</code></td>
<td>Times SQE test error was reported.</td>
</tr>
</tbody>
</table>
macxmt_errors  Packets encountering transmit MAC failures, except carrier and collision failures.
macrcv_errors  Packets received with MAC errors, except align, fcs, and tooolong errors.
toolong_errors  Packets received larger than the maximum permitted length.
runt_errors  Packets received smaller than the minimum permitted length.

The following group of statistics applies to networks of type DL_TPR; these are maintained by device-specific drivers of that type, as above.

line_errors  Packets received with non-data bits or FCS errors.
burst_errors  Times an absence of transitions for five half-bit timers was detected.
signal_losses  Times loss of signal condition on the ring was detected.
ace_errors  Times an AMP or SMP frame in which A is equal to C is equal to 0, was followed by another such SMP frame without an intervening AMP frame.
internal_errors  Times the station recognized an internal error.
lost_frame_errors  Times the TRR timer expired during transmit.
frame_copied_errors  Times a frame addressed to this station was received with the FS field A bit set to 1.
token_errors  Times the station acting as the active monitor recognized an error condition that needed a token transmitted.
freq_errors  Times the frequency of the incoming signal differed from the expected frequency.

The following group of statistics applies to networks of type DL_FDDI; these are maintained by device-specific drivers of that type, as above.

mac_errors  Frames detected in error by this MAC that had not been detected in error by another MAC.
mac_lost_errors  Frames received with format errors such that the frame was stripped.
mac_tokens  Number of tokens received (total of non-restricted and restricted).
mac_tvx_expired  Number of times that TVX has expired.

gld(7D)

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

mac_late
Number of TRT expirations since this MAC was reset or a token was received.

mac_ring_ops
Number of times the ring has entered the "Ring_Operational" state from the "Ring Not Operational" state.

FILES
/kernel/misc/gld loadable kernel module

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

Contrary to the DLPI specification, GLD returns the device’s correct address length and broadcast address in DL_INFO_ACK even before the stream has been attached to a PPA.

Promiscuous mode may only be entered by Streams that are attached to a PPA.
The glm Host Bus Adapter driver is a SCSA compliant nexus driver that supports the LSI 53c810, LSI 53c875, LSI 53c876, LSI 53C896 and LSI 53C1010 SCSI chips. It supports the standard functions provided by the SCSA interface. That is, it supports tagged and untagged queuing, Narrow/Wide/Fast/Ultra SCSI/Ultra SCSI 2/Ultra SCSI 3, and auto request sense, but it does not support linked commands.

Configure the glm driver by defining properties in glm.conf. These properties override the global SCSI settings. glm supports these properties which can be modified by the user: scsi-options, target<n>-scsi-options, scsi-reset-delay, scsi-tag-age-limit, scsi-watchdog-tick, and scsi-initiator-id.

target<n>-scsi-options overrides the scsi-options property value for target<n>. <n> can vary from decimal 0 to 15. glm supports these scsi-options: SCSI_OPTIONS_DR, SCSI_OPTIONS_SYNC, SCSI_OPTIONS_TAG, SCSI_OPTIONS_FAST, SCSI_OPTIONS_WIDE, SCSI_OPTIONS_FAST20, SCSI_OPTIONS_FAST40 and SCSI_OPTIONS_FAST80.

After periodic interval scsi-watchdog-tick, the glm driver searches through all current and disconnected commands for timeouts.

scsi-tag-age-limit is the number of times that the glm driver attempts to allocate a particular tag ID that is currently in use after going through all tag IDs in a circular fashion. After finding the same tag ID in use scsi-tag-age-limit times, no more commands will be submitted to this target until all outstanding commands complete or timeout.

Refer to scsi_hbaAttach(9F).

EXAMPLE 1 Using the glm Configuration File

Create a file called /kernel/drv(glm.conf and add the following line:

```
scsi-options=0x78;
```

This disables tagged queuing, Fast/Ultra SCSI and wide mode for all glm instances.

The following example disables an option for one specific glm (refer to driver.conf(4) and pci(4) for more details):

```
name="glm" parent="/pci@1f,4000"
  unit-address="3"
  target1-scsi-options=0x58
  scsi-options=0x178 scsi-initiator-id=6;
```

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.
EXAMPLE 1 Using the glm Configuration File  (Continued)

The example above sets scsi-options for target 1 to 0x58 and all other targets on this SCSI bus to 0x178.

The physical pathname of the parent can be determined using the /devices tree or following the link of the logical device name:

```
# ls -l /dev/rdsk/c0t0d0s0
lrwxrwxrwx 1 root root 45 May 16 10:08 /dev/rdsk/c0t0d0s0 -> .. / .. /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

In this case, like the example above, the parent is /pci@1f,4000 and the unit-address is the number bound to the scsi@3 node.

To set scsi-options more specifically per target:

```
target1-scsi-options=0x78;
device-type-scsi-options-list = "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

The above sets scsi-options for target 1 to 0x78 and for all other targets on this SCSI bus to 0x3f8 except for one specific disk type which will have scsi-options set to 0x58.

scsi-options specified per target ID have the highest precedence, followed by scsi-options per device type. Global scsi-options (for all glm instances) per bus have the lowest precedence.

The system needs to be rebooted before the specified scsi-options take effect.

Driver Capabilities

The target driver needs to set capabilities in the glm driver in order to enable some driver features. The target driver can query and modify these capabilities: synchronous, tagged-qing, wide-xfer, auto-rgsense, qfull-retries, qfull-retry-interval. All other capabilities can only be queried.

By default, tagged-qing, auto-rgsense, 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.

```plaintext
FILES
/kernel/drvglm   ELF Kernel Module
/kernel/drvglm.conf  Optional configuration file
```

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

```plaintext
ATTRIBUTE TYPE | ATTRIBUTE VALUE
--- | ---
Architecture | Limited to PCI-based systems with LSI 53c810, LSI 53c875, LSI 53c876, LSI 53c896 and LSI 53c1010 SCSI I/O processors
```

```plaintext
SEE ALSO
prtconf(1M), driver.conf(4), pci(4), attributes(5), scsi_abort(9F), scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F), scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)
```

```plaintext
Writing Device Drivers

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

LSI Logi Inc (formerly Symbios Logic Inc.):

- SYM53c810 PCI-SCSI I/O processor with Narrow operation
- SYM53c875 PCI-SCSI I/O Processor With Fast-20
- SYM53c876 PCI-SCSI I/O processor Dual channel Fast-20
- SYM53c896 PCI-SCSI I/O processor Dual channel Fast-40
- SYM53c1010 PCI-SCSI I/O processor Dual Channel Fast-80
```

```plaintext
DIAGNOSTICS
The messages described below are some that may appear on the system console, as well as being logged.

Device is using a hilevel intr
The device was configured with an interrupt level that cannot be used with this glm driver. Check the PCI device.

map setup failed
Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

glm_script_alloc failed
The driver was unable to load the SCRIPTS for the SCSI processor, check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

cannot map configuration space.
The driver was unable to map in the configuration registers. Check for bad hardware. SCSI devices will be inaccessible.

 glm(7D)
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.
The glm driver supports the following LSI chips:

- LSI 53C810, which supports Narrow, Fast SCSI mode. The maximum SCSI bandwidth is 10 MB/sec.
- LSI 53C875, which supports Wide, Fast, and Ultra SCSI mode. The maximum SCSI bandwidth is 40 MB/sec.
- LSI 53C896, which supports Wide, Fast and Ultra SCSI 2 mode. The maximum LVD SCSI bandwidth is 80 MB/sec.
- LSI 53c1010, which supports wide, Fast and Ultra SCSI 3 mode. The maximum LVD SCSI bandwidth is 160 MB/sec.

The glm driver exports properties indicating per target the negotiated transfer speed (target<n>-sync-speed), whether wide bus is supported (target<n>-wide), for that particular target (target<n>-scsi-options), and whether tagged queuing has been enabled (target<n>-TQ). The sync-speed property value is the data transfer rate in KB/sec. The target<n>-TQ and the target<n>-wide property have value 1 to indicate that the corresponding capability is enabled, or 0 to indicate that the capability is disabled for that target. Refer to *prtconf*(1M) (verbose option) for viewing the glm properties.

scsi, instance #0
Driver properties:
```
    name <target6-TQ> length <4>
      value <0x00000000>.
    name <target6-wide> length <4>
      value <0x00000000>.
    name <target6-sync-speed> length <4>
      value <0x00002710>.
    name <target1-TQ> length <4>
      value <0x00000001>.
    name <target1-wide> length <4>
      value <0x00000000>.
    name <target1-sync-speed> length <4>
      value <0x00002710>.
    name <target0-TQ> length <4>
      value <0x00000001>.
    name <target0-wide> length <4>
      value <0x00000001>.
    name <target0-sync-speed> length <4>
      value <0x00009c40>.
    name <scsi-options> length <4>
      value <0x000007f8>.
    name <scsi-watchdog-tick> length <4>
      value <0x0000000a>.
    name <scsi-tag-age-limit> length <4>
      value <0x00000002>.
    name <scsi-reset-delay> length <4>
      value <0x00000bb8>.
    name <latency-timer> length <4>
      value <0x00000088>.
    name <cache-line-size> length <4>
      value <0x00000010>.
```
gpio_87317(7D)

NAME      gpio_87317 – General purpose I/O driver for SuperIO

DESCRIPTION The gpio_87317 driver is the general purpose I/O driver for the National Semiconductor SuperIO (PC87317) chipset. It supports remote system controller (RSC) administration via an interface to the SuperIO’s general purpose I/O bits.

FILES       /kernel/drv/sparcv9/gpio_87317
            64-bit 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>Limited to SPARC systems with SuperIO</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcarx.u</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
</tbody>
</table>

SEE ALSO            PC87317VUL/PC97317VUL SuperI/O Data Sheet — National Semiconductor
NAME  hci1394 – 1394 OpenHCI host controller driver

SYNOPSIS  firewire@unit-address

DESCRIPTION  The hci1394 host controller driver is an IEEE 1394 compliant nexus driver that supports the 1394 Open Host Controller Interface Specification 1.0, an industry standard developed by Sun, Apple, Compaq, Intel, Microsoft, National Semiconductor, and Texas Instruments. The hci1394 driver supports asynchronous transfers, isochronous transfers, and bus reset management. The hci1394 driver also supports the nexus device control interface.

FILES  /kernel/drv/sparcv9/hci1394
       64-bit ELF kernel module

ATTRIBUTES  See attributes(5) for a description 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 PCI-based SPARC systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNW1394x</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
</tbody>
</table>

SEE ALSO  IEEE 1394 - IEEE Standard for a High Performance Serial Bus

1394 Open Host Controller Interface Specification 1.0
hdio(7I)

NAME
hdio – SMD and IPI disk control operations

SYNOPSIS
#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.

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

/ *
  * Used for drive info
  */
struct hdk_type {
  ushort_t hdkt_hsect; /* hard sector count (read only) */
  ushort_t hdkt_promrev; /* prom revision (read only) */
  uchar_t hdkt_drtype; /* drive type (ctlr specific) */
  uchar_t hdkt_drstat; /* drive status (ctlr specific, ro) */
};

HDKIOCGBAD The argument is a pointer to a hdk_badmap structure (described below). This ioctl() is used to get the bad sector map from the disk.

HDKIOCSBAD The argument is a pointer to a hdk_badmap structure (described below). This ioctl() is used to set the bad sector map on the disk.

/ *
  * Used for bad sector map
  */
struct hdk_badmap {
  caddr_t hdkb_bufaddr; /* address of user’s map buffer */
};

HDKIOCGDIAG The argument is a pointer to a hdk_diag structure (described below). This ioctl() gets the most recent command that failed along with the sector and error number from the hard disk.

/ *
  * Used for disk diagnostics
  */
struct hdk_diag {
  ushort_t hdkd_errcmd; /* most recent command in error */
  daddr_t hdkd_errsect; /* most recent sector in error */
  uchar_t hdkd_errno; /* most recent error number */
  uchar_t hdkd_severe; /* severity of most recent error */
};
SEE ALSO

ioctl(2), dkio(7I), xd(7D), xy(7D)
hid – Human interface device (HID) class driver

keyboard@unit-address
mouse@unit-address

The hid driver is a USBa (Solaris USB Architecture) compliant client driver that supports the Human Interface Device Class (HID) 1.0 specification. The Human Interface Device (HID) class encompasses devices controlled by humans to operate computer systems. Typical examples of HID devices include keyboards, mice, trackballs, and joysticks. HID also covers front-panel controls such as knobs, switches, and buttons. A USB device with multiple interfaces may have one interface for audio and a HID interface to define the buttons that control the audio.

The hid driver is general and primarily handles the USB functionality of the device and generic HID functionality. For example, HID interfaces are required to have an interrupt pipe for the device to send data packets, and the hid driver opens the pipe to the interrupt endpoint and starts polling. The hid driver is also responsible for managing the device through the default control pipe. In addition to being a USB client driver, the hid driver is also a STREAMS driver so that modules may be pushed on top of it.

The HID specification is flexible, and HID devices dynamically describe their packets and other parameters through the HID report descriptor. The HID parser is a misc module that parses the HID report descriptor and creates a database of information about the device. The hid driver queries the HID parser to find out the type and characteristics of the HID device. The HID specification predefines packet formats for the boot protocol keyboard and mouse.

FILES
/kernel/drv/hid
  32 bit ELF kernel module
/kernel/drv/sparcv9/hid
  64 bit ELF kernel module
/kernel/misc/hidparser
  /kernel/misc/sparcv9/hidparser

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 PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO
hubd(7D), ohci(7D), uhci(7D), usb_mid(7D), usbkbm(7M), usbms(7M), usba(7D)

Writing Device Drivers
STREAMS Programming Guide

204  man pages section 7: Device and Network Interfaces • Last Revised 10 Oct 2000
Diagnostics  

<table>
<thead>
<tr>
<th>Universal Serial Bus Specification 1.0 and 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Class Definition for Human Interface Devices (HID) 1.0</td>
</tr>
<tr>
<td>System Administration Guide: Basic Administration</td>
</tr>
</tbody>
</table>

**Diagnostics**  
None.

**Notes**  
The hid driver currently supports only keyboards and mice.
NAME
hme – SUNW, hme Fast-Ethernet device driver

SYNOPSIS
/dev/hme

DESCRIPTION
The SUNW, hme Fast-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS
hardware driver supporting the connectionless Data Link Provider Interface,
dlpi(7P), over a SUNW, hme Fast-Ethernet controller. The motherboard and add-in
SBus SUNW, hme controllers of several varieties are supported. Multiple SUNW, hme
controllers installed within the system are supported by the driver.

The hme driver provides basic support for the SUNW, hme hardware. It is used to
handle the SUNW, hme device. Functions include chip initialization, frame transit and
receive, multicast and promiscuous support, and error recovery and reporting.
SUNW, hme The SUNW, hme device provides 100Base-TX networking interfaces using
SUN’s FEPS ASIC and an Internal Transceiver. The FEPS ASIC provides the Sbus
interface and MAC functions and the Physical layer functions are provided by the
Internal Transceiver which connects to a RJ-45 connector. In addition to the RJ-45
connector, an MII (Media Independent Interface) connector is also provided on all
SUNW, hme devices except the SunSwith SBus adapter board. The MII interface is used
to connect to an External Transceiver which may use any physical media (copper or
fiber) specified in the 100Base-TX standard. When an External Transceiver is connected
to the MII, the driver selects the External Transceiver and disables the Internal
Transceiver.

The 100Base-TX standard specifies an “auto-negotiation” protocol to automatically
select the mode and speed of operation. The Internal transceiver is capable of doing
“auto-negotiation” with the remote-end of the link (Link Partner) and receives the
capabilities of the remote end. It selects the Highest Common Denominator mode
of operation based on the priorities. It also supports forced-mode of operation
where the driver can select the mode of operation.

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

hme and DLPI

The hme driver is a “style 2” Data Link Service provider. All M_PROTO and
M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives
are defined in <sys/dlpi.h>. Refer to dlpi(7P) for more information. An explicit
DL_ATTACH_REQ message by the user is required to associate the opened stream with
a particular device (ppa). The ppa ID is interpreted as an unsigned long data type
and indicates the corresponding device instance (unit) number. An error
(DL_ERROR_ACK) is returned by the driver if the ppa field value does not correspond
to a valid device instance number for this system. The device is initialized on first
attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to the
DL_INFO_REQ from the user are as follows:

■ The maximum SDU is 1500 (ETHERMTU - defined in <sys/ethernet.h> ).
■ The minimum SDU is 0.
The dlsap address length is 8.
The MAC type is DL_ETHER.
The sap length values is −2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
The service mode is DL_CLDLS.
No optional quality of service (QOS) support is included at present so the QOS fields are 0.
The provider style is DL_STYLE2.
The version is DL_VERSION_2.
The broadcast address value is Ethernet/IEEE broadcast address (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
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_PROMISCOFF_REQ and DL_PROMISC_PHYS primitives with the DL_PROMISC_MULTI flag set in the dl_level field enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive in that it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

hme DRIVER

By default, the hme driver performs “auto-negotiation” to select the mode and speed of the link, when the Internal Transceiver is used.

When an External Transceiver is connected to the MII interface, the driver selects the External Transceiver for networking operations. If the External Transceiver supports “auto-negotiation”, the driver uses the auto-negotiation procedure to select the link speed and mode. If the External Transceiver does not support auto-negotiation, it will select the highest priority mode supported by the transceiver.

- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex
The link can be in one of the 4 following modes:

These speeds and modes are described in the 100Base-TX standard.

The *auto-negotiation* protocol automatically selects:

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

The auto-negotiation protocol does the following:

- Gets all the modes of operation supported by the Link Partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities

The *internal transceiver* is capable of all of the operating speeds and modes listed above. When the internal transceiver is used, by default, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the Link Partner.

When an *external transceiver* is connected to the MII interface, the driver selects the external transceiver for networking operations. If the external transceiver supports auto-negotiation:

- The driver uses the auto-negotiation procedure to select the link speed and mode.

If the external transceiver *does not* support auto-negotiation

- The driver selects the highest priority mode supported by the transceiver.

Sometimes, the user may want to select the speed and mode of the link. The SUNW, hme device supports programmable “IPG” (Inter-Packet Gap) parameters ipg1 and ipg2. By default, the driver sets ipg1 to 8 byte-times and ipg2 to 4 byte-times (which are the standard values). Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mbps and accordingly, IPG will be set to 9.6 or 0.96 microseconds.

The hme driver provides for setting and getting various parameters for the SUNW, hme device. The parameter list includes:

<table>
<thead>
<tr>
<th>Parameter List</th>
</tr>
</thead>
<tbody>
<tr>
<td>current transceiver status</td>
</tr>
<tr>
<td>current link status</td>
</tr>
<tr>
<td>inter-packet gap</td>
</tr>
<tr>
<td>local transceiver capabilities</td>
</tr>
<tr>
<td>link partner capabilities</td>
</tr>
</tbody>
</table>

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
hme(7D)

(RW) capabilities. At boot time, these two sets of capabilities will be the same. The Link Partner capabilities are also read only parameters because the current default value of these parameters can only be read and cannot be modified.

FILES
/dev/hme hme special character device
/kernel/drv/hme.conf System-wide default device driver properties

SEE ALSO
ndd(1M), netstat(1M), driver.conf(4), dlpi(7P), le(7D)
The hpfc fibre channel host bus adapter is a SCSA compliant nexus driver that supports all Agilent fibre channel host bus adapters, including the HHBA5100x, HHBA5101x, and HHBA5121x models. Agilent host bus adapters support the fibre channel protocol on private fibre channel arbitrated loops and fabrics. The driver supports up to ten host bus adapters, with a maximum of 125 fibre channel devices on each host bus adapter. The hpfc driver supports a maximum of 256 LUNs per target.

The hpfc driver does not support the BIOS Int 13 feature, which enables the booting of an operating system. As a result, you should not install an operating system on devices attached to the hpfc driver.

The hpfc driver attempts to configure itself using the information in the /kernel/drv/hpfc.conf configuration file.

By default, the driver supports only LUN 0 for each target device. To add multiple LUN support, modify the /kernel/drv/sd.conf file.

Before upgrading the hpfc driver, backup the sd.conf file to save customized LUN settings and then use pkgrm(1M) to remove the old version of the driver.

The host bus adapter port is initialized to FL_Port when connected to a fabric switch. To change it to F_Port, add the init_as_nport=1 entry to the hpfc.conf file and reboot the system.

To conserve system resources, at least one disk drive must be attached to the hpfc driver. If no devices are attached, the driver will not load.

FILES
/kernel/drv/hpfc 32-bit ELF kernel module
/kernel/drv/sparcv9/hpfc 64-bit ELF kernel module
/kernel/drv/hpfc.conf Driver configuration file
/kernel/drv/sd.conf SCSI disk configuration file

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

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

SEE ALSO
luxadm(1M), pkgrm(1M), prtconf(1M), driver.conf(4), attributes(5), ses(7D), ssd(7D)

ANSI X3.272–1996, Fibre Channel Arbitrated Loop (FC-AL),
ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP),
ANSI X3.270-1996, SCSI-3 Architecture Model (SAM),
Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)
HSFS is a file system 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 file system 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 file system also contains support for the Rock Ridge Extensions. If the extensions are contained on the CD-ROM, then the file system will provide all of the file system semantics and file types of UFS, except for writability and hard links.

If your /etc/vfstab file contains a line similar to

/dev/dsk/c0t6d0s0 -/hsfs hsfs -no ro and /hsfs exists, you can mount an HSFS file system with either of the following commands:

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

```
mount /hsfs
```

Normally, if Rock Ridge extensions exist on the CD-ROM, the file system 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 file system; 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 support only regular files and directories, thus HSFS supports only 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.

**EXAMPLE 1** Sample Display of File System Files

If there is a file `BIG.BAR` on a High Sierra or ISO 9660 format CD-ROM it will show up as `big.bar` when listed on a HSFS file system.

If there are three files

```
BAR.BAZ;1
BAR.BAZ;2
and
BAR.BAZ;3
```

on a High Sierra or ISO 9660 format CD-ROM, only the file `BAR.BAZ;3` will be accessible. It will be listed as `bar.baz`.

**SEE ALSO** `mount(1M)`, `mount_hsfs(1M)`, `vfstab(4)`


N. V. Phillips and Sony Corporation, *System Description of Compact Disc Read Only Memory*, ("Yellow Book").


**DIAGNOSTICS**

`hsfs`: Warning: the file system... does not conform to the ISO-9660 spec

The specific reason appears on the following line. You might be attempting to mount a CD-ROM containing a different file system, such as UFS.

`hsfs`: Warning: the file system... contains a file [with an] unsupported type

The `hsfs` file system does not support the format of some file or directory on the CD-ROM, for example a record structured file.

`hsfs`: hsnode table full, %d nodes allocated

There are not enough HSFS internal data structure elements to handle all the files currently open. This problem may be overcome by adding a line of the form `set hsfs:nhsnode=number` to the `/etc/system` system configuration file and rebooting. See `system(4)`.
Do not physically eject a CD-ROM while the device is still mounted as a HSFS file system.

Under MS-DOS (for which CD-ROMs are frequently targeted), files with no extension may be represented either as

```
filename
```

or

```
filename.
```

that is, with or without a trailing period. These names are not equivalent under UNIX systems. For example, the names

```
BAR
```

and

```
BAR.
```

are not names for the same file under the UNIX system. This may cause confusion if you are consulting documentation for CD-ROMs originally intended for MS-DOS systems.

Use of the `-onotraildot` option to `mount(1M)` makes it optional to specify the trailing dot. (See `mount_hsfs(1M)`).

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.
hubd(7D)

NAME  hubd – USB hub driver

SYNOPSIS  hub@unit-address

DESCRIPTION  The hubd is a USBA (Solaris USB Architecture) compliant client driver that supports USB hubs conforming to the *Universal Serial Bus Specification* 1.0 and 1.1. The hubd driver supports bus–powered and self–powered hubs. The driver supports hubs with individual port power, ganged power and no power switching.

When a device is attached to the hub port, the hubd driver enumerates the devices by determining the type of device and assigning an address to it. The hubd driver also attaches a driver to the device if one is available. When the device is disconnected from the hub port, the hubd driver offlines any driver instance attached to the device.

FILES  /kernel/drv/hubd
   32 bit ELF kernel module

   /kernel/drv/sparcv9/hubd
   64 bit ELF kernel module

ATTRIBUTES  See attributes(5) for a description 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 PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO  ohci(7D), uhci(7D), usba(7D), usb_mid(7D)

*Writing Device Drivers*

*Universal Serial Bus Specification* 1.0 and 1.1

*System Administration Guide: Basic Administration*

DIAGNOSTICS  The messages described below may appear on the system console as well as being logged. All messages are formatted in the following manner:

WARNING: <device path> <usb<instance number>>: Error message...

where <instance number> is the instance number of hubd and <device path> is the physical path to the device in /devices directory. Messages from the root hub are displayed with a usb<instance number> prefix instead of hub<instance number> as the root hub is an integrated part of the host controller.

Connecting device on port <number> failed.

The driver failed to enumerate device connected on port <number> of hub. If enumeration fails, you should disconnect and re-connect.
Global over current condition. Please disconnect.

The driver detected an over current condition. This means that the aggregate current being drawn by the devices on the downstream port exceeds a preset value. Refer to section 7.2.1.2.1 and 11.13.5 of the Universal Serial Bus Specification 1.1. You must remove and insert this hub to render it and its downstream devices functional again. If this message continues to display for a particular hub, you may need to remove downstream devices to eliminate the problem.

Cannot access device. Please reconnect <device name>.

This hub has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Devices not identical to the previous one on this port.
Please disconnect and reconnect.

Same condition as described above; however in this case, the driver is unable to identify the original device with a name string.

Local power has been lost, please disconnect hub.

The USB self-powered hub has lost external power. All USB devices connected downstream to this hub will cease to function. Disconnect the hub, plug in the external power-supply and then plug in the hub again.

Hub driver supports max of <n> ports on hub.
Hence, using the first <number of physical ports> of <n> ports available.

The current hub driver supports hubs that have <n> ports or less. A hub with more than <n> ports has been plugged in. Only the first <n> out of the total <number of physical ports> ports are usable.
The I2O Block Storage OSM abstraction (BSA, which also is referred to as block storage class) layer is the primary interface that Solaris operating environments 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 IA for disk. The maximum size disk supported by `i2o_bs` is the same as what is available on IA.

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/pci@0,0/pci101e,0@10,1/disk@10:a
```

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.

```
/dev/dsk/cndn[s|p]n
/dev/rdsk/cndn[s|p]n
```
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>IA</td>
</tr>
</tbody>
</table>

**SEE ALSO**
fdisk(1M), format(1M), mount(1M), lseek(2), read(2), write(2), readdir(3C),

vfstab(4), acct(3HEAD), attributes(5), dkio(7I)
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.

The configuration file for the `i2o_scsi` driver is `/kernel/drv/i2o_scsi.conf`. There are no user-configurable options in this file.

### Attributes

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

### SEE ALSO

`attributes(5)`

*Solaris 8 (Intel Platform Edition) Installation Guide*

### NOTES

Throughout the release, support of additional devices may be added. See the *Solaris 8 (Intel Platform Edition) 2/02 Hardware Compatibility List* for additional information.
The ICMP6 protocol is the error and control message protocol used with Version 6 of the Internet Protocol. It is used by the kernel to handle and report errors in protocol processing. It is also used for IPv6 neighbor and router discovery, and for multicast group membership queries and reports. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a “raw socket” type is used. The protocol number for ICMP6, used in the `proto` parameter to the socket call, can be obtained from `getprotobyname(3SOCKET)`. ICMP6 file descriptors and sockets are connectionless and are normally used with the `t_sndudata` / `t_rcvudata` and the `sendto()` / `recvfrom()` calls. They may also be used with the `sendmsg()` / `recvmsg()` calls when sending or receiving ancillary data.

Outgoing packets automatically have an Internet Protocol Version 6 (IPv6) header and zero or more IPv6 extension headers prepended. These headers are prepended by the kernel. Unlike ICMP for IPv4, the `IP_HDRINCL` option is not supported for ICMP6, so ICMP6 applications neither build their own outbound IPv6 headers, nor do they receive the inbound IPv6 headers with received data. IPv6 extension headers and relevant fields of the IPv6 header may be set or received as ancillary data to a `sendmsg(3SOCKET)` or `recvmsg(3SOCKET)` system call. Each of these fields and extension headers may also be set on a per socket basis with the `setsockopt(3SOCKET)` system call. Such "sticky" options are used on all outgoing packets unless overridden by ancillary data. When any ancillary data is present with a `sendmsg(3SOCKET)` system call, all sticky options are ignored for that system call, but subsequently remain configured.

ICMP6 is a datagram protocol layered above IPv6. Received ICMP6 messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of each ICMP6 error message received by the system is provided to every holder of an open ICMP6 socket or TLI descriptor.

**SEE ALSO**
`getprotobyname(3SOCKET)`, `recv(3SOCKET)`, `recvmsg(3SOCKET)`, `send(3SOCKET)`, `sendmsg(3SOCKET)`, `setsockopt(3SOCKET)`,
`t_rcvudata(3NSL), t_sndudata(3NSL), inet6(7P), ip6(7P), routing(7P)`


**DIAGNOSTICS**
A socket operation may fail with one of the following errors returned:
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EISCONN</td>
<td>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.</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.</td>
</tr>
<tr>
<td>ENOBUFS</td>
<td>The system ran out of memory for an internal data structure.</td>
</tr>
<tr>
<td>EADDRNOTAVAIL</td>
<td>An attempt was made to create a socket with a network address for which no network interface exists.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The system was unable to allocate memory for an internal data structure.</td>
</tr>
<tr>
<td>ENOPROTOOPT</td>
<td>An attempt was made to set an IPv4 socket option on an IPv6 socket.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>An attempt was made to set an invalid or malformed socket option.</td>
</tr>
<tr>
<td>EAFNOSUPPORT</td>
<td>An attempt was made to bind or connect to an IPv4 or mapped address, or to specify an IPv4 or mapped address as the next hop.</td>
</tr>
</tbody>
</table>
icmp(7P)

NAME
icmp, ICMP – Internet Control Message Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>
s = socket(AF_INET, SOCK_RAW, proto);
t = t_open("/dev/icmp", O_RDWR);

DESCRIPTION
ICMP is the error and control message protocol used by the Internet protocol family. It is
used by the kernel to handle and report errors in protocol processing. It may also be
accessed by programs using the socket interface or the Transport Level Interface
(“TLI”) for network monitoring and diagnostic functions. When used with the socket
interface, a “raw socket” type is used. The protocol number for ICMP, used in the proto
parameter to the socket call, can be obtained from getprotobyname(3SOCKET).
ICMP file descriptors and sockets are connectionless, and are normally used with the
t_sndudata / t_rcvudata and the sendto() / recvfrom() calls.

Outgoing packets automatically have an Internet Protocol (“IP”) header prepended to
them. Incoming packets are provided to the user with the IP header and options intact.

ICMP is a 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(3SOCKET), recv(3SOCKET), send(3SOCKET),
t_rcvudata(3NSL), t_sndudata(3NSL), inet(7P), ip(7P), routing(7P)

Postel, Jon, Internet Control Message Protocol — DARPA Internet Program Protocol
Specification, RFC 792, Network Information Center, SRI International, Menlo Park,
Calif., September 1981.

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:

EINVAL
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.

ENOBUSFS
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.
idn – inter-domain network device driver

/dev/idn

The idn driver is a multi-thread, loadable, clonable, STREAMS-based pseudo driver that supports the connectionless Data Link Provider Interface dlpi(7P) over the Sun Enterprise 10000 Gigaplane-XB Interconnect. This connection is permitted only between domains within the same Sun Enterprise 10000 server.

The idn driver supports 1 to 32 logical network interfaces that can be connected to domains linked to the local domain through the domain_link(1M) command. (See domain_link(1M) in the Sun Enterprise 10000 SSP 3.2 Reference Manual for more information.) The idn driver works in conjunction with the System Service Processor (SSP) to perform domain linking/unlinking and automated linking upon host bootup.

The /dev/idn device is used to access all IDN services provided by the system.

IDN and DLPI

The idn driver is a style-2 Data Link Service provider. All M_PROTO and M_PCPROTO–type messages are interpreted as DLPI primitives. For the idn driver to associate the opened stream with a particular device (ppa), you must send an explicit DL_ATTACH_REQ message. The ppa ID is interpreted as an unsigned long and indicates the corresponding device instance (unit) number. The DL_ERROR_ACK error is returned by the driver if the ppa field value does not correspond to a valid device-instance number for the system. The device is initialized on first attach and de-initialized (stopped) on the last detach.

- The maximum SDU is configurable by using the idn.conf file and has a range of 512 bytes to 512 Kbytes. The default value is 16384 bytes.
- The minimum SDU is 0.
- The Service Access Pointer (SAP) address length is 8.
- The MAC type is DL_ETHER.
- The SAP length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- The service mode is DL_CLDLS.
- Optional quality of service (QOS) is not presently supported; accordingly, the QOS fields are 0.
- The provider style is DL_STYLE2.
- The version is DL_VERSION_2.
- The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFFF). The idn driver supports broadcast by issuing messages to each target individually. The idn driver is inherently a point-to-point network between domains. When the idn driver is in the DL_ATTACHED state, the user must send a DL_BIND_REQ request to associate a particular SAP with the stream. The idn driver interprets the SAP field within the DL_BIND_REQ message as an Ethernet type and valid values for the SAP field are in the range of 0 to 0xFFFF. Only one Ethernet type can be bound to the stream at any time.
If a SAP with a value of 0 is selected, the receiver will be in 802.3 mode. All frames received from the media having a type field in the range of 0 to 1500 are assumed to be 802.3 frames and are routed up all open streams which are bound to SAP value 0. If more than one stream is in 802.3 mode, then the frame will be duplicated and routed up as multiple stream DL_UNITDATA_IND messages.

In transmission, the driver checks the SAP field of the DL_BIND_REQ to determine if the SAP value is 0, and if the destination type field is in the range of 0 to 1500. If either is true, the driver computes the length of the message, (excluding the initial message block m_proto mb1k) of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The driver also supports raw M_DATA mode. When the user sends a DLIOCRAW ioctl, the particular stream is put in raw mode. A complete frame and a proper ether header is expected as part of the data.

The DLSAP address format consists of the 6-byte, physical address component (Ethernet) followed immediately by the 2-byte SAP component (type), producing an 8-byte DLSAP address. Applications should not hardcode to this particular implementation-specific DLSAP address format, but instead should use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The SAP length, full DLSAP length, and SAP physical ordering are included within the DL_INFO_ACK primitive. The physical address length can be computed by subtracting the SAP length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ message to obtain the current physical address associated with the stream.

When the idn driver is in the DL_BOUND state, you can transmit frames on the IDN by sending DL_UNITDATA_REQ messages to the driver. The driver then routes received IDN frames up the open and bound streams having a SAP which matches the Ethernet type as DL_UNITDATA_IND messages. If necessary, received IDN frames are duplicated and routed up multiple open streams. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the SAP (type) and physical (Ethernet) components.

**IDN Primitives**

In addition to the mandatory connectionless DLPI message set, the idn driver supports the following primitives:

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives which enable or disable, respectively, the reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following the DL_ATTACHED state.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives, which with the DL_PROMISC_PHYS flag set in the dl_level field, enable or disable, respectively, the reception of all promiscuous frames on the media, including frames generated by the local domain. When used with the DL_PROMISC_SAP flag set in the dl_level field,
these primitives enable or disable, respectively, the reception of all SAP (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set in the dl_level field, these primitives enable or disable, respectively, the reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other SAP and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive which returns the 6-octet, Ethernet address associated with (or attached to) the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ request.

Because the driver maintains domain address information in the address to direct packets to the correct destination, the DL_SET_PHYS_ADDR_REQ primitive is not allowed.

FILES The following files are supported:
/dev/idn
   IDN special character device
/platform/SUNW,Ultra-Enterprise-10000/kernel/drv/idn.conf
   System-wide and per-interface default device driver properties

SEE ALSO netstat(1M), ndd(1M), dlpi(7P)

   domain_link(1M) in the Sun Enterprise 10000 SSP 3.2 Reference Manual.

   Sun Enterprise 10000 InterDomain Networks User Guide

NOTES The idn driver supports a set of properties that can be set by using the driver.conf file for the IDN. See the Sun Enterprise 10000 InterDomain Networks User Guide for more information about the properties in the driver.conf(4), (idn.conf, for IDNs).
### NAME
ifb – IFB graphics accelerator driver

### DESCRIPTION
The ifb driver is the device driver for the Sun Elite3D graphics accelerators. The ifbdaemon process loads the ifb microcode at system startup time and during the resume sequence of a suspend-resume cycle.

### FILES
- `/dev/fbs/ifb`
  - Device special file
- `/usr/lib/ifb.ucode`
  - ifb microcode
- `/usr/sbin/ifbdaemon`
  - ifb microcode loader

### SEE ALSO
- SUNWifb_config(1M)
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.

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

Cannot alloc fcal handle
   Driver was unable allocate a dma handle for retrieving loop map from the Host Bus
Adapter. Driver did not attach to device; SCSI devices will be inaccessible.

Cannot bind portdb
   Driver was unable to bind fcal port handle to the memory used for obtaining port
database. Driver did not attach to device; SCSI devices will be inaccessible.

scsi_hba_attach failed
   Driver was unable to attach to the SCSA framework. Driver did not attach to
device; SCSI devices will be inaccessible.

Unable to create hotplug thread
   Driver was not able to create the kernel thread used for hotplug support. Driver did
not attach to device; SCSI devices will be inaccessible.
Cannot add intr
   Driver was not able to add the interrupt routine to the kernel. Driver did not attach
to device; SCSI devices will be inaccessible.

Unable to attach
   Driver was unable to attach to the hardware for some reason that may be printed.
   Driver did not attach to device; SCSI devices will be inaccessible.

The following set of messages may be display at any time. They will be printed with
the full device pathname followed by the shorter form described above.

Firmware checksum incorrect
   Firmware has an invalid checksum and will not be downloaded.

Chip reset timeout
   ISP chip failed to reset in the time allocated; may be bad hardware.

Stop firmware failed
   Stopping the firmware failed; may be bad hardware.

Load ram failed
   Unable to download new firmware into the ISP chip.

DMA setup failed
   The DMA setup failed in the host adapter driver on a scsi_pkt. This will return
   TRAN_BADPKT to a SCSA target driver.

Bad request pkt type
Bad request pkt
Bad request pkt hdr
Bad req pkt order
   The ISP Firmware rejected the packet as being set up incorrectly. This will cause the
   ifp driver to call the target completion routine with the reason of CMD_TRAN_ERR
   set in the scsi_pkt. Check the target driver for correctly setting up the packet.

Firmware error
   The ISP chip encountered a firmware error of some kind. This error will cause the
   ifp driver to do error recovery by resetting the chip.

DMA Failure (event)
   The ISP chip encountered a DMA error while reading from the request queue
   (event is 8003) or writing to the response queue (event is 8004). This error will cause
   the ifp driver to do error recovery by resetting the chip.

Fatal error, resetting interface
   This is an indication that the ifp driver is doing error recovery. This will cause all
   outstanding commands that have been transported to the ifp driver to be
   completed via the scsi_pkt completion routine in the target driver with reason of
   CMD_RESET and status of STAT_BUS_RESET set in the scsi_pkt.
target t, duplicate port wwns
   The driver detected target t to be having the same port WWN as a different target;
   this is not supposed to happen. Target t will become inaccessible.

target t, duplicate switch settings
   The driver detected devices with the same switch setting t. All such devices will
   become inaccessible.

WWN changed on target t
   The World Wide Name (WWN) has changed on the device with switch setting t.

target t, unknown device type dt
   The driver does not know the device type dt reported by the device with switch
   setting t.
if_tcp(7P)

NAME

if_tcp, if – general properties of Internet Protocol network interfaces

DESCRIPTION

A network interface is a device for sending and receiving packets on a network. It is usually a hardware device, although it can be implemented in software. Network interfaces used by the Internet Protocol (IPv4 or IPv6) must be STREAMS devices conforming to the Datalink Provider Interface (DLPI). See dlpi(7P).

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)), and the SIOCSLIFNAME ioctl(2) is issued to specify the name of the interface and whether it is IPv4 or IPv6. 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 SIOCSLIFADDR ioctl() before it can be used. On interfaces where the network-to-link layer address mapping is static, only the network number is taken from the ioctl() request; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-to-link layer address mapping facilities, for example, 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.

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

```c
/* Interface request structure used for socket ioctl. All */
/* interface ioctl must have parameter definitions which */
/* begin with ifr_name. The remainder may be interface specific. */
struct lifreq {
  #define LIFNAMSIZ 32
  char  lfr_name[LIFNAMSIZ]; /* if name, for example "le1" */
  union {
    int    lifru_addrlen; /* for subnet/token etc */
    uint_t lifru_ppa; /* SIOCSLIFNAME */
  } lifr_lifru1;
  union {
    struct sockaddr_storage lifru_addr;
    struct sockaddr_storage lifru_dstaddr;
    struct sockaddr_storage lifru_broadaddr;
    struct sockaddr_storage lifru_token; /* With lifr_addrlen */
    struct sockaddr_storage lifru_subnet; /* With lifr_addrlen */
    int    lifru_index; /* interface index */
    uint64_t lifru_flags; /* SIOC?LIFFLAGS */
  } lifr_lifru;
  uint_t lifru_data[1]; /* interface dependent data */
  char    lifru_enaddr[6];
  int     lifr_muxid[2]; /* mux id’s for arp and ip */
  struct lif_ifinfo_req lifru_ifinfo_req;
  struct lif_nd_req  lifru_nd_req;
} lifr_lifru;
```

#define lifr_addrlen lifr_lifru1.lifru_addrlen
#define lifr_ppa lifr_lifru1.lifru_ppa /* Driver’s ppa */
### if_tcp(7P)

```c
#define lifr_addr lifr_lifru.lifru_addr     /* address */
#define lifr_dstaddr lifr_lifru.lifru_dstaddr
#define lifr_broadaddr lifr_lifru.lifru_broadaddr /* broadcast address */
#define lifr_token lifr_lifru.lifru_token    /* address token */
#define lifr_subnet lifr_lifru.lifru_subnet   /* subnet prefix */
#define lifr_flags lifr_lifru.lifru_flags     /* flags */
#define lifr_metric lifr_lifru.lifru_metric   /* metric */
#define lifr_index lifr_lifru.lifru_index     /* mtu */
#define lifr_ip_muxid lifr_lifru.lifru_ip_muxid
#define lifr_arp_muxid lifr_lifru.lifru_arp_muxid
#define lifr_nd lifr_lifru.lifru_nd             /* SIOCLIF*ND */
#define lifr_ifinfo lifr_lifru.lifru_ifinfo    /* SIOC[GS]LIFLNKINFO */
};
```

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

#### SIOCGLIFADDR
Get interface address.

#### SIOCSLIFDSTADDR
Set point to point address for interface.

#### SIOCGLIFDSTADDR
Get point to point address for interface.

#### SIOCSLIFFLAGS
Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

#### SIOCGLIFFLAGS
Get interface flags.

#### SIOCSLIFCONF
Get interface configuration list. This request takes an lifconf structure (see below) as a value-result parameter. The lifc_len field should be initially set to the size of the buffer pointed to by lifc_buf. On return it will contain the length, in bytes, of the configuration list. The lifc_family field should be set to AF_UNSPEC to retrieve both AF_INET and AF_INET6 interfaces. The lifc_flags field should be initially set to zero.

#### SIOCGLIFNUM
Get number of interfaces. This request returns an integer which is the number of interface descriptions (struct lifreq) that will be returned by the SIOCGLIFCONF ioctl; that is, it gives an indication of how large lifc_len has to be. This request takes an lifnum structure (see below) as a value-result parameter. The lifn_family field should be set to AF_UNSPEC to count both AF_INET and AF_INET6 interfaces. The lifn_flags field should be initially set to zero.
SIOCGLIFMTU

Set the maximum transmission unit (MTU) size for interface. Place the result of this request in ifru_mtu field. The MTU can not exceed the physical MTU limitation (which is reported in the DLPI DL_INFO_ACK message).

SIOCGLIFMTU

Get the maximum transmission unit size for interface. Place the result of this request in ifru_mtu field.

SIOCGLIFMETRIC

Set the metric associated with the interface. The metric is used by routine daemons such as in.routed(1M).

SIOCGLIFMETRIC

Get the metric associated with the interface.

SIOCGLIFMUXID

Get the ip and arp muxid associated with the interface.

SIOCGLIFMUXID

Set the ip and arp muxid associated with the interface.

SIOCGLIFINDEX

Get the interface index associated with the interface.

SIOCGLIFINDEX

Set the interface index associated with the interface.

SIOCLIFADDIF

Add a new logical interface on a physical interface using an unused logical unit number.

SIOCLIFREMOVEIF

Remove a logical interface by specifying its IP address or logical interface name.

SIOCSLIFTOKEN

Set the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.

SIOCSLIFTOKEN

Get the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.

SIOCSLIFSUBNET

Set the subnet prefix associated with the interface.

SIOCSLIFSUBNET

Get the subnet prefix associated with the interface.

SIOCGLIFLNKINFO

Set link specific parameters for the interface.

SIOCGLIFLNKINFO

Get link specific parameters for the interface.

SIOCLIFDELND

Delete a neighbor cache entry for IPv6.

SIOCLIFGETND

Get a neighbor cache entry for IPv6.

SIOCLIFSETND

Set a neighbor cache entry for IPv6.

SIOCTMYADDR

Test if the address is assigned to this node. This request takes an sioc_addrreq structure (see below) as a value-result parameter. The sa_addr field should be set to the address to test. The sa_res field will contain a non-zero value if the address is assigned to this node.
SIOCTONLINK

Test if the address is directly reachable, for example, that it can be reached without going through a router. This request takes an `sioc_addrreq` structure (see below) as a value-result parameter. The `sa_addr` field should be set to the address to test. The `sa_res` field will contain a non-zero value if the address is onlink.

SIOCTMYSITE

Test if the address is part of the same site as this node. This request takes an `sioc_addrreq` structure (see below) as a value-result parameter. The `sa_addr` field should be set to the address to test. The `sa_res` field will contain a non-zero value if the address is in the same site.

The `lifconf` structure has the form:

```c
/*
 * Structure used in SIOCGLIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct lifconf {
    sa_family_t lifc_family;
    int lifc_flags; /* request specific interfaces */
    int lifc_len; /* size of associated buffer */
    union {
        caddr_t lifcu_buf;
        struct ifreq *lifcu_req;
    } lifc_lifcu;

#define lifc_buf lifc_lifcu.lifcu_buf /* buffer address */
#define lifc_req lifc_lifcu.lifcu_req /* array of structures returned */
};
```

The `sioc_addrreq` structure has the form:

```c
/* Structure used in SIOCGLIFNUM request. */
struct lifnum {
    sa_family_t lifn_family;
    int lifn_flags; /* request specific interfaces */
    int lifn_count; /* Result */
};

/*
 * Argument structure for SIOCT* address testing ioctls.
 */
struct sioc_addrreq {
    struct sockaddr_storage sa_addr; /* Address to test */
    int sa_res; /* Result - 0/1 */
};
```
The following ioctl() calls are maintained for compatibility but only apply to IPv4 network interfaces, since the data structures are too small to hold an IPv6 address. Unless specified otherwise, the request takes an ifreq structure as its parameter. This structure has the form:

```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 */
    /* *le1* */
    union {
        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 */
        char ifru_enaddr[6];
        int ifru_index; /* interface index */
        int if_muxid[2]; /* mux id's for arp and ip */
    } ifr_ifru;

    #define ifr_addr ifr_ifru.ifru_addr /* address */
    #define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of p-to-p link */
    #define ifr_oname ifr_ifru.ifru_oname /* other if name */
    #define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
    #define ifr_flags ifr_ifru.ifru_flags /* flags */
    #define ifr_index ifr_ifru.ifru_index /* interface index */
    #define ifr_metric ifr_ifru.ifru_metric /* metric */
    #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.

**SIOCSIFMETRIC**
Set the metric associated with the interface. The metric is used by routine daemons such as in.routed(1M).

**SIOCGIFMETRIC**
Get the metric associated with the interface.

**SIOCGIFMUXID**
Get the ip and arpmuxid associated with the interface.

**SIOCSIFMUXID**
Set the ip and arpmuxid associated with the interface.

**SIOCGIFINDEX**
Get the interface index associated with the interface.

**SIOCSIFINDEX**
Set the interface index associated with the interface.

The *ifconf* structure has the form:

```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;
        struct ifreq *ifcu_req;
    } ifc_ifcu;

#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};
```

**ERRORS**
EPERM The effective user id of the calling process in not superuser.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENXIO</td>
<td>The <code>lifr_name</code> member of the <code>lifreq</code> structure contains an invalid value.</td>
</tr>
<tr>
<td>EBADADDR</td>
<td>Wrong address family or malformed address.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>For <code>SIOCSLIFFLAGS</code>, this error is returned when the order of bringing the</td>
</tr>
<tr>
<td></td>
<td>primary/physical interface (for example, <code>le0</code>) and a secondary/logical</td>
</tr>
<tr>
<td></td>
<td>interface associated with the same physical interface (for example, <code>le0:1</code>)</td>
</tr>
<tr>
<td></td>
<td>up or down is violated. The physical interface must be configured up first</td>
</tr>
<tr>
<td></td>
<td>and cannot be configured down until all the corresponding logical interfaces</td>
</tr>
<tr>
<td></td>
<td>have been configured down.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>For <code>SIOCGLIFCONF</code>, this error is returned when the size of the buffer</td>
</tr>
<tr>
<td></td>
<td>pointed to by the <code>lifc_buf</code> member of the <code>lifconf</code> structure is too small.</td>
</tr>
<tr>
<td></td>
<td>For <code>SIOCGLIFMTU</code>, this error is returned when the requested MTU size is</td>
</tr>
<tr>
<td></td>
<td>invalid. This error indicates the MTU size is greater than the MTU size</td>
</tr>
<tr>
<td></td>
<td>supported by the DLPI provider or less than 68 (for IPv4) or less than 1200</td>
</tr>
<tr>
<td></td>
<td>(for IPv6).</td>
</tr>
</tbody>
</table>

**SEE ALSO**
- `ifconfig(1M)`, `in.routed(1M)`, `ioctl(2)`, `arp(7P)`, `dlpi(7P)`, `ip(7P)`, `ip6(7P)`, `streamio(7I)`
inet6(7P)

NAME
inet6 – Internet protocol family for Internet Protocol version 6

SYNOPSIS
#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION
The inet6 protocol family implements a collection of protocols that are centered around the Internet Protocol version 6 (IPv6) and share a common address format. The inet6 protocol family can be accessed using the socket interface, where it supports the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types, or the Transport Level Interface (TLI), where it supports the connectionless (T_CLTS) and connection oriented (T_COTS_ORD) service types.

PROTOCOLS
The Internet protocol family for IPv6 included the Internet Protocol Version 6 (IPv6), the Neighbor Discovery Protocol (NDP), the Internet Control Message Protocol (ICMPv6), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP supports the socket interface’s SOCK_STREAM abstraction and TLI’s T_COTS_ORD service type. UDP supports the SOCK_DGRAM socket abstraction and the TLI T_CLTS service type. See tcp(7P) and udp(7P). A direct interface to IPv6 is available using the socket interface. See ip6(7P). ICMPv6 is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs. See icmp6(7P). NDP is used to translate 128-bit IPv6 addresses into 48-bit Ethernet addresses.

IPv6 addresses come in three types: unicast, anycast, and multicast. A unicast address is an identifier for a single network interface. An anycast address is an identifier for a set of interfaces; a packet sent to an anycast address is delivered to the "nearest" interface identified by that address, pursuant to the routing protocol’s measure of distance. A multicast address is an identifier for a set of interfaces; a packet sent to a multicast address is delivered to all interfaces identified by that address. There are no broadcast addresses as such in IPv6; their functionality is superseded by multicast addresses.

For IPv6 addresses, there are three scopes within which unicast addresses are guaranteed to be unique. The scope is indicated by the address prefix. The three varieties are link-local (the address is unique on that physical link), site-local (the address is unique within that site), and global (the address is globally unique).

The three highest order bits for global unicast addresses are set to 001. The ten highest order bits for site-local addresses are set to 1111 1110 11. The ten highest order bits for link-local addresses are set to 1111 1110 11. For multicast addresses, the eight highest order bits are set to 1111 1111. Anycast addresses have the same format as unicast addresses.

IPv6 addresses do not follow the concept of "address class" seen in IP.

A global unicast address is divided into the following segments:

- The first three bits are the Format Prefix identifying a unicast address.
The next 13 bits are the Top-Level Aggregation (TLA) identifier. For example, the identifier could specify the ISP.

The next eight bits are reserved for future use.

The next 24 bits are the Next-Level Aggregation (NLA) identifier.

The next 16 bits are the Site-Level Aggregation (SLA) identifier.

The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

Link-local unicast addresses are divided in this manner:

- The first ten bits are the Format Prefix identifying a link-local address.
- The next 54 bits are zero.
- The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

Site-local unicast addresses are divided in this manner:

- The first ten bits are the Format Prefix identifying a site-local address.
- The next 38 bits are zero.
- The next 16 bits are the subnet ID.
- The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

IPv6 addresses are sixteen byte quantities, stored in network byte order. The socket API uses the `sockaddr_in6` structure when passing IPv6 addresses between an application and the kernel. The `sockaddr_in6` structure has the following members:

- `sa_family_t sin6_family;`
- `in_port_t sin6_port;`
- `uint32_t sin6_flowinfo;`
- `struct in6_addr sin6_addr;`
- `uint32_t sin6_scope_id;`
- `uint32_t __sin6_src_id;`

Library routines are provided to manipulate structures of this form. See `inet(3SOCKET)`.

The `sin6_addr` field of the `sockaddr_in6` structure specifies a local or remote IPv6 address. Each network interface has one or more IPv6 addresses configured, that is, a link-local address, a site-local address, and one or more global unicast IPv6 addresses.

The special value of all zeros may be used on this field to test for "wildcard" matching. Given in a `bind(3SOCKET)` call, this value leaves the local IPv6 address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IPv6 addresses of the system. This can prove useful when a process neither knows nor cares what the local IPv6 address is, or when a process wishes to receive requests using all of its network interfaces. The `sockaddr_in6` structure given in the `bind()` call must specify an `in6_addr` value of either all zeros or one of the system’s valid IPv6 addresses. Requests to bind any other address will elicit the error.
When a connect(3SOCKET) call is made for a socket that has a wildcard local address, the system sets the sin6_addr field of the socket to the IPv6 address of the network interface through which the packets for that connection are routed.

The sin6_port field of the sockaddr_in6 structure specifies a port number used by TCP or UDP. The local port address specified in a bind() call is restricted to be greater than 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 cannot be in use by any socket of the same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, the system picks a unique port address greater than IPPROTO_RESERVED. A unique local port address is also selected when a socket which is not bound is used in a connect(3SOCKET) or sendto() call. See send(3SOCKET). This allows programs that do not care which local port number is used to set up TCP connections by simply calling socket(3SOCKET) and then connect(3SOCKET), and then sending UDP datagrams with a socket() call followed by a sendto() call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IPv6 addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the SO_REUSEADDR socket option with setsockopt(). See getsockopt(3SOCKET).

In addition, the same port may be bound by two separate sockets if one is an IP socket and the other an IPv6 socket.

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used using the TLI.

SEE ALSO
ioctl(2), bind(3SOCKET), connect(3SOCKET), getipnodebyaddr(3SOCKET), getipnodebyname(3SOCKET), getprotobyname(3SOCKET), getprotobynumber(3SOCKET), getprotobynumber(3SOCKET), getservbyname(3SOCKET), getsockbyaddr(3SOCKET), inet(3SOCKET), ip(7P), ip6(7P), tcp(7P), udp(7P)


NOTES
The IPv6 support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.
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.

The Internet protocol family is comprised of the Internet Protocol ("IP"), the Address Resolution Protocol ("ARP"), the Internet Control Message Protocol ("ICMP"), the Transmission Control Protocol ("TCP"), and the User Datagram Protocol ("UDP").

TCP supports the socket interface’s SOCK_STREAM abstraction and TLI’s T_COTS_ORD service type. UDP supports the SOCK_DGRAM socket abstraction and the TLI T_CLTS service type. See tcp(7P) and udp(7P). A direct interface to IP is available using both TLI and the socket interface (see ip(7P)). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs (see icmp(7P)). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses (see arp(7P)).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded. The most-significant bit is zero in Class A addresses, in which the high-order 8 bits represent the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of IP networks may choose 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.

IP addresses are four byte quantities, stored in network byte order. IP addresses should be manipulated using the byte order conversion routines (see byteorder(3SOCKET)).
Addresses in the Internet protocol family use the `sockaddr_in` structure, which has the 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(3SOCKET)`.

The `sin_addr` field of the `sockaddr_in` structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value `INADDR_ANY` may be used in this field to effect "wildcard" matching. Given in a `bind(3SOCKET)` call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The `sockaddr_in` structure given in the `bind(3SOCKET)` call must specify an `in_addr` value of either `INADDR_ANY` or one of the system's valid IP addresses. Requests to bind any other address will elicit the error `EADDRNOTAVAIL`. When a `connect(3SOCKET)` call is made for a socket that has a wildcard local address, the system sets the `sin_addr` field of the socket to the IP address of the network interface that the packets for that connection are routed through.

The `sin_port` field of the `sockaddr_in` structure specifies a port number used by TCP or UDP. The local port address specified in a `bind(3SOCKET)` call is restricted to be greater than `IPPORT_RESERVED` (defined in `<netinet/in.h>`), unless the creating process is running as the superuser, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error `EADDRINUSE`. If the local port address is specified as 0, then the system picks a unique port address greater than `IPPORT_RESERVED`. A unique local port address is also picked when a socket which is not bound is used in a `connect(3SOCKET)` or `sendto` (see `send(3SOCKET)`) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling `socket(3SOCKET)` and then `connect(3SOCKET)`, and to send UDP datagrams with a `socket(3SOCKET)` call followed by a `sendto()` call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the `SO_REUSEADDR` socket option with `setsockopt` (see `getsockopt(3SOCKET)`).

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used using the TLI.
getservbyname(3SOCKET), getsockopt(3SOCKET), send(3SOCKET),
socket(3SOCKET), arp(7P), icmp(7P), ip(7P), tcp(7P), udp(7P)

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information

**NOTES**
The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.
**NAME**
ip6 – Internet Protocol Version 6

**SYNOPSIS**
```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip6.h>

s = socket(AF_INET6, SOCK_RAW, proto);
t = t_open("/dev/rawip6", O_RDWR);
```

**DESCRIPTION**
The IPv6 protocol is the next generation of the internetwork datagram delivery protocol of the Internet protocol family. Programs may use IPv6 through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IPv6. See `tcp(7P)` and `udp(7P)`. Direct access may be by means of the socket interface, using a "raw socket," or by means of the Transport Level Interface (TLI). The protocol options and IPv6 extension headers defined in the IPv6 specification may be set in outgoing datagrams.

The STREAMS driver `/dev/rawip6` is the TLI transport provider that provides raw access to IPv6.

Raw IPv6 sockets are connectionless and are normally used with the `sendto()` and `recvfrom()` calls (see `send(3SOCKET)` and `recv(3SOCKET)`), although the `connect(3SOCKET)` call may also be used to fix the destination for future datagrams. In this case, the `read(2)` or `recv(3SOCKET)` and `write(2)` or `send(3SOCKET)` calls may be used. Ancillary data may also be sent or received over raw IPv6 sockets using the `sendmsg(3SOCKET)` and `recvmsg(3SOCKET)` system calls.

Unlike raw IP, IPv6 applications do not include a complete IPv6 header when sending; there is no IPv6 analog to the IP `IP_HDRINCL` socket option. IPv6 header values may be specified or received as ancillary data to a `sendmsg(3SOCKET)` or `recvmsg(3SOCKET)` system call, or may be specified as "sticky" options on a per-socket basis by using the `setsockopt(3SOCKET)` system call. Such sticky options are applied to all outbound packets unless overridden by ancillary data. If any ancillary data is specified in a `sendmsg(3SOCKET)` call, all sticky options not explicitly overridden revert to default values for that datagram only; the sticky options persist as set for subsequent datagrams.

Since `sendmsg(3SOCKET)` is not supported for `SOCK_STREAM` upper level protocols such as TCP, ancillary data is unsupported for TCP. Sticky options, however, are supported.

Since `sendmsg(3SOCKET)` is supported for `SOCK_DGRAM` upper level protocols, both ancillary data and sticky options are supported for UDP, ICMP6, and raw IPv6 sockets.

The socket options supported at the IPv6 level are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_BOUND_IF</td>
<td>Limit reception transmission of packets to this interface. Takes an integer as an argument; the integer is the selected interface index.</td>
</tr>
</tbody>
</table>
### IPV6_UNSPEC_SRC
Boolean. Allow/disallow sending with a zero source address.

### IPV6_UNICAST_HOPS
Default hop limit for unicast datagrams. This option takes an integer as an argument. Its value becomes the new default value for `ip6_hops` that IPv6 will use on outgoing unicast datagrams sent from that socket. The initial default is 60.

### IPV6_CHECKSUM
Specify the integer offset in bytes into the user data of the checksum location. Does not apply to the ICMP6 protocol. Note: checksums are required for all IPv6 datagrams; this is different from IP, in which datagram checksums were optional. IPv6 will compute the ULP checksum if the value in the checksum field is zero.

The following options are boolean switches controlling the reception of ancillary data:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_RECV_PKTINFO</td>
<td>Enable/disable receipt of the index of the interface the packet arrived on, and of the inbound packet’s destination address.</td>
</tr>
<tr>
<td>IPV6_RECV_HOPLIMIT</td>
<td>Enable/disable receipt of the inbound packet’s current hop limit.</td>
</tr>
<tr>
<td>IPV6_RECV_HOPOPTS</td>
<td>Enable/disable receipt of the inbound packet’s IPv6 hop-by-hop extension header.</td>
</tr>
<tr>
<td>IPV6_RECV_DSTOPTS</td>
<td>Enable/disable receipt of the inbound packet’s IPv6 destination options extension header.</td>
</tr>
<tr>
<td>IPV6_RECV_ROUTHDR</td>
<td>Enable/disable receipt of the inbound packet’s IPv6 routing header.</td>
</tr>
<tr>
<td>IPV6_RECV_RT_HDR_DSTOPTS</td>
<td>Enable/disable receipt of the inbound packet’s intermediate-hops options extension header.</td>
</tr>
</tbody>
</table>

The following options may be set as sticky options with `setsockopt(3SOCKET)` or as ancillary data to a `sendmsg(3SOCKET)` system call:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_PKTINFO</td>
<td>Set the source address and/or interface out which the packet(s) will be sent. Takes a <code>struct ip6_pktinfo</code> as the parameter.</td>
</tr>
<tr>
<td>IPV6_HOPLIMIT</td>
<td>Set the initial hop limit for outbound datagrams. Takes an integer as the parameter. Note: This option sets the hoplimit only for ancillary data or sticky options and does not change the default hoplimit for the socket; see <code>IPV6_UNICAST_HOPS</code> and <code>IPV6_MULTICAST_HOPS</code> to change the socket’s default hoplimit.</td>
</tr>
<tr>
<td>IPV6_NEXTHOP</td>
<td>Specify the IPv6 address of the first hop, which must be a neighbor of the sending host. Takes a <code>struct</code></td>
</tr>
</tbody>
</table>
sockaddr_in6 as the parameter. When this option specifies the same address as the destination IPv6 address of the datagram, this is equivalent to the existing SO_DONTROUTE option.

**IPV6_HPOPTS**
Specify one or more hop-by-hop options. Variable length. Takes a complete IPv6 hop-by-hop options extension header as the parameter.

**IPV6_DSTOPTS**
Specify one or more destination options. Variable length. Takes a complete IPv6 destination options extension header as the parameter.

**IPV6_RTHDR**
Specify the IPv6 routing header. Variable length. Takes a complete IPv6 routing header as the parameter. Currently, only type 0 routing headers are supported.

**IPV6_RTHRDSTOPTS**
Specify one or more destination options for all intermediate hops. May be configured, but will not be applied unless an IPv6 routing header is also configured. Variable length. Takes a complete IPv6 destination options extension header as the parameter.

The following options affect the socket’s multicast behavior:

**IPV6_JOIN_GROUP**
Join a multicast group. Takes a struct ipv6_mreq as the parameter; the structure contains a multicast address and an interface index.

**IPV6_LEAVE_GROUP**
Leave a multicast group. Takes a struct ipv6_mreq as the parameter; the structure contains a multicast address and an interface index.

**IPV6_MULTICAST_IF**
The outgoing interface for multicast packets. This option takes an integer as an argument; the integer is the interface index of the selected interface.

**IPV6_MULTICAST_HOPS**
Default hop limit for multicast datagrams. This option takes an integer as an argument. Its value becomes the new default value for ip6_hops that IPv6 will use on outgoing multicast datagrams sent from that socket. The initial default is 1.

**IPV6_MULTICAST_LOOP**
Loopback for multicast datagrams. Normally multicast datagrams are delivered to members on the sending host. Setting the unsigned character argument to 0 will cause the opposite behavior.

The multicast socket options can be used with any datagram socket type in the IPv6 family.
At the socket level, the socket option `SO_DONTROUTE` may be applied. This option forces datagrams being sent to bypass routing and forwarding by forcing the IPv6 hoplimit field to 1, meaning that the packet will not be forwarded by routers.

Raw IPv6 datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IPv6 layer in two directions: from the network up to user processes and from user processes down to the network. Using this orientation, IPv6 is layered above the network interface drivers and below the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) is logically a part of IPv6. See `icmp6(7P)`.

Unlike IP, IPv6 provides no checksum of the IPv6 header. Also unlike IP, upper level protocol checksums are required. IPv6 will compute the ULP/data portion checksum if the checksum field contains a zero (see `IPV6_CHECKSUM` option above).

IPv6 extension headers in received datagrams are processed in the IPv6 layer according to the protocol specification. Currently recognized IPv6 extension headers include hop-by-hop options header, destination options header, routing header (currently, only type 0 routing headers are supported), and fragment header.

The IPv6 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 IPv6 interfaces that are up. This behavior can be overridden by using `ndd(1M)` to set the `/dev/ip6` variable, `ip6_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 and whether or not the neighbor discovery protocol daemon configuration file `/etc/inet/ndpd.conf` exists. The default value is zero; `ip6_forwarding` is set to 1 only if more than one interface has been configured for IPv6 and if `/etc/inet/ndpd.conf` exists.

The IPv6 layer will send an ICMP6 message back to the source host in many cases when it receives a datagram that can not be handled. A "time exceeded" ICMP6 message will be sent if the `ip6_hops` field in the IPv6 header drops to zero in the process of forwarding a datagram. A "destination unreachable" message will be sent by a router or by the originating host if a datagram can not be sent on because there is no route to the final destination; it will be sent by a router when it encounters a firewall prohibition; it will be sent by a destination node when the transport protocol (that is, TCP) has no listener. A "packet too big" message will be sent by a router if the packet is larger than the MTU of the outgoing link (this is used for Path MTU Discovery). A "parameter problem" message will be sent if there is a problem with a field in the IPv6 header or any of the IPv6 extension headers such that the packet cannot be fully processed.

The IPv6 layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the...
network interface. Fragments of received datagrams are dropped from the reassembly
queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IPv6
back up to the user process.

SEE ALSO
ndd(1M), read(2), write(2), bind(3SOCKET), connect(3SOCKET),
getsockopt(3SOCKET), recv(3SOCKET), recvmsg(3SOCKET), send(3SOCKET),
sendmsg(3SOCKET), setsockopt(3SOCKET), defaultrouter(4), icmp6(7P),
if_tcp(7P), inet6(7P), routing(7P) tcp(7P), udp(7P)

Deering, S. and Hinden, B., Internet Protocol, Version 6 (IPv6) Specification, RFC 2460,
Copyright The Internet Society (C) 1998, December, 1998.

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.

EADDRNOTAVAIL A bind() operation was attempted for an address that
is not configured on this machine.

EINVAL A sendmsg() operation with a non-NULL
msg_accrights was attempted.

EINVAL A getsockopt() or setsockopt() operation with
an unknown socket option name was given.

EINVAL A getsockopt() or setsockopt() operation was
attempted with the IPv6 option field improperly
formed; an option field was shorter than the minimum
value or longer than the option buffer provided; the
value in the option field was invalid.

EISCONN A connect() operation was attempted on a socket on
which a connect() operation had already been
performed, and the socket could not be successfully
disconnected before making the new connection.

EISCONN A sendto() or sendmsg() operation specifying an
address to which the message should be sent was
attempted on a socket on which a connect()  
operation had already been performed.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSGSIZE</td>
<td>A <code>send()</code>, <code>sendto()</code>, or <code>sendmsg()</code> operation was attempted to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>An attempt was made to establish a connection via <code>connect()</code>, or to send a datagram via <code>sendto()</code> or <code>sendmsg()</code>, where there was no matching entry in the routing table; or if an ICMP “destination unreachable” message was received.</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>A <code>send()</code> or <code>write()</code> operation, or a <code>sendto()</code> or <code>sendmsg()</code> operation not specifying an address to which the message should be sent, was attempted on a socket on which a <code>connect()</code> operation had not already been performed.</td>
</tr>
<tr>
<td>ENOBUFS</td>
<td>The system ran out of memory for fragmentation buffers or other internal data structures.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The system was unable to allocate memory for an IPv6 socket option or other internal data structures.</td>
</tr>
<tr>
<td>ENOPROTOOPT</td>
<td>An IP socket option was attempted on an IPv6 socket, or an IPv6 socket option was attempted on an IP socket.</td>
</tr>
</tbody>
</table>

**NOTES** Applications using the sockets API must use the Advanced Sockets API for IPv6 (RFC 2292) to see elements of the inbound packet’s IPv6 header or extension headers.
### NAME
ip, IP – Internet Protocol

### SYNOPSIS
```
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, proto);
t = t_open (*/dev/rawip*, O_RDWR);
```

### DESCRIPTION
IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IP. See tcp(7P) and udp(7P). Direct access may be by means of the socket interface, using a “raw socket,” or by means of the Transport Level Interface (“TLI”). The protocol options defined in the IP specification may be set in outgoing datagrams.

The STREAMS driver /dev/rawip is the TLI transport provider that provides raw access to IP.

Raw IP sockets are connectionless and are normally used with the sendto() and recvfrom() calls (see send(3SOCKET) and recv(3SOCKET)), although the connect(3SOCKET) call may also be used to fix the destination for future datagram. In this case, the read(2) or recv(3SOCKET) and write(2) or send(3SOCKET) calls may be used. If proto is IPPROTO_RAW or IPPROTO_IGMP, the application is expected to include a complete IP header when sending. Otherwise, that protocol number will be set in outgoing datagrams and used to filter incoming datagrams and an IP header will be generated and prepended to each outgoing datagram. In either case, received datagrams are returned with the IP header and options intact.

The socket options supported at the IP level are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_OPTIONS</td>
<td>IP options for outgoing datagrams. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with setsockopt() (see getsockopt(3SOCKET)). The getsockopt(3SOCKET) call returns the IP options set in the last setsockopt() call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in setsockopt() matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IP_SEC_OPT</strong></td>
<td>Enable or obtain IPsec security settings for this socket.</td>
</tr>
<tr>
<td></td>
<td>For more details on the protection services of IPsec, see ipsec(7P).</td>
</tr>
<tr>
<td><strong>IP_ADD_MEMBERSHIP</strong></td>
<td>Join a multicast group.</td>
</tr>
<tr>
<td><strong>IP_DROP_MEMBERSHIP</strong></td>
<td>Leave a multicast group.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_MULTICAST_IF</strong></td>
<td>The outgoing interface for multicast packets. This option takes a <code>struct in_addr</code> as an argument, and it selects that interface for outgoing IP multicast packets. If the address specified is <strong>INADDR_ANY</strong>, it will use the unicast routing table to select the outgoing interface (which is the default behavior).</td>
</tr>
<tr>
<td><strong>IP_MULTICAST_TTL</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>IP_MULTICAST_LOOP</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>IP_TOS</strong></td>
<td>This option takes an integer argument as its input value. The least significant 8 bits of the value are used to set the Type Of Service field in the IP header of the outgoing packets.</td>
</tr>
</tbody>
</table>

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.

Additionally, finer-grained forwarding can be configured in IP. Each interface will create an <ifname>:ip_forwarding /dev/ip variable that can be modified using ndd(1M). If a per-interface :ip_forwarding variable is set to 0, packets will neither be forwarded from this interface to others, nor forwarded to this interface. Setting the ip_forwarding variable will toggle all of the per-interface :ip_forwarding variables to the setting of ip_forwarding.

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(3SOCKET), connect(3SOCKET),
getsockopt(3SOCKET), recv(3SOCKET), send(3SOCKET), defaultrouter(4),
icmp(7P), if_tcp(7P), inet(7P), ip6(7P), ipsec(7P), routing(7P), tcp(7P),
udp(7P)

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

Postel, J., RFC 791, Internet Protocol – DARPA Internet Program Protocol Specification,
Information Sciences Institute, University of Southern California, September 1981.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

EACCES A bind() operation was attempted with a “reserved” port number and the effective user ID of the process was not the privileged user.

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

EADDRNOTAVAIL A bind() operation was attempted for an address that is not configured on this machine.

EINVAL A sendmsg() operation with a non-NULL msg_accrights was attempted.

EINVAL A getsockopt() or setsockopt() operation with an unknown socket option name was given.

EINVAL A getsockopt() or setsockopt() operation was attempted with the IP option field improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

EISCONN A connect() operation was attempted on a socket on which a connect() operation had already been performed, and the socket could not be successfully disconnected before making the new connection.

EISCONN A sendto() or sendmsg() operation specifying an address to which the message should be sent was attempted on a socket on which a connect() operation had already been performed.

EMSGSIZE A send(), sendto(), or sendmsg() operation was attempted to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
### ip(7P)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENETUNREACH</td>
<td>An attempt was made to establish a connection via <code>connect()</code> or to send a datagram via <code>sendto()</code> or <code>sendmsg()</code>, where there was no matching entry in the routing table; or if an ICMP “destination unreachable” message was received.</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>A <code>send()</code> or <code>write()</code> operation, or a <code>sendto()</code> or <code>sendmsg()</code> operation not specifying an address to which the message should be sent, was attempted on a socket on which a <code>connect()</code> operation had not already been performed.</td>
</tr>
<tr>
<td>ENOBUFS</td>
<td>The system ran out of memory for fragmentation buffers or other internal data structures.</td>
</tr>
</tbody>
</table>

### 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, 82558, 82559–controlled network interface controllers

### SYNOPSIS
/dev/iprb

### DESCRIPTION
The iprb Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over Intel D100 82557, 82558, and 82559 controllers. Multiple 82557, 82558, and 82559 controllers installed within the system are supported by the driver. The iprb driver provides basic support for the 82557, 82558, and 82559 hardware. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting.

The cloning, character-special device /dev/iprb is used to access all 82557, 82558, and 82559 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 primitives supported by the driver.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are as follows:

- Maximum SDU is 1500 (ETHERMTU).
- Minimum SDU is 0. The driver will pad to the mandatory 60-octet minimum packet size.
- The dlsap address length is 8.
- MAC type is DL_ETHER.
- The sap length value is −2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

### Known Problems and Limitations
IA based systems with the Intel EtherExpress PRO/100B or the Intel EtherExpress PRO/100+ might hang when the interface is brought down at the very instant that a packet is being received. To avoid this, wait until the system is experiencing light or no network traffic before bringing the interface down.

Early versions of the firmware on Intel EtherExpress PRO/100+ and Intel PRO/100+ Management adapters do not support PXE network boot on Solaris systems. Upgrade the firmware if the version is lower than 078. PXE firmware versions are expressed as three-digit build numbers. The build number is typically displayed by the firmware during boot. If the PXE build number is not displayed during boot, change the system BIOS or adapter BIOS configuration to display PXE messages during boot.

### FILES
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iprb</td>
<td>Device special file</td>
</tr>
<tr>
<td>/kernel/drv/iprb.conf</td>
<td>iprb configuration file</td>
</tr>
<tr>
<td>&lt;sys/stropts.h&gt;</td>
<td>stropts network header file</td>
</tr>
</tbody>
</table>
iprb(7D)

<sys/ethernet.h> Ethernet network header file
<sys/dlpi.h> dlpi network header file
<sys/gld.h> gld network header file

The iprb.conf configuration file options include:

-TxURRetry
   Default: 3
   Allowed Values: 0, 1, 2, 3
   Sets the number of retransmissions. Modified when tuning performance.

-MWIEnable
   Default: 0 (Disable)
   Allowed Values: 0 (Disable), 1 (Enable)
   Should only be set for 82558 adapters and systems in which the PCI bus supports
   Memory Write & Invalidate operations. Can improve the performance for some
   configurations.

-FlowControl
   Default: 0 (Disable)
   Allowed Values: 0 (Disable), 1 (Enable)
   Setting this value can improve the performance for some configurations.

-CollisionBackOffModification
   Default: 0 (Disable)
   Allowed Values: 0 (Disable), 1 (Enable)
   Setting this value can improve the performance for some configurations.

-PhyErrataFrequency
   Default: 0 (Disable)
   Allowed Values: 0 (Disable), 10 (Enable)
   If you have problems establishing links with cables length = 70 Ft, set this field to
   10

-CpuCycleSaver
   Default: 0
   Allowed Values: 1 through FFFFh
   Reasonable Values: 200h through 800h
The CPUSaver algorithm improves the system’s P/E ratio by reducing the number of interrupts generated by the card. The algorithm bundles multiple receive frames together, then generates a single interrupt for the bundle. Because the microcode does not support run-time configuration, configuration must be done prior to the microcode being loaded into the chip. Changing this value from its default means that the driver will have to be unloaded and loaded for the change to take affect. Setting the CpuCycleSaver option to 0 prevents the algorithm from being used. Because it varies for different network environments, the optimal value for this parameter is impossible to predict. Accordingly, developers should run tests to determine the effect that changing this value has on bandwidth and CPU utilization.

-ForceSpeedDuplex
  Default: 5 (Auto-negotiate)

  Allowed Values: 4 (100 FDX)
  3 (100 HDX)
  2 (10 FDX)
  1 (10 HDX)

  Specify the speed and duplex mode for each instance.

  Example: ForceSpeedDuplex=5,4;

  Sets iprb0 to autonegotiate and iprb1 to 100 FDX.

Attributes

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

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

See Also

attributes(5), dlpi(7P), gld(7D)
### NAME
ipsec – Internet Protocol Security Architecture

### DESCRIPTION
The IP Security Architecture (IPsec) provides protection for IP datagrams. The protection can include confidentiality, strong integrity of the data, partial sequence integrity (replay protection), and data authentication. IPsec is performed inside the IP processing, and it can be applied with or without the knowledge of an Internet application.

IPsec provides two mechanisms for protecting data. The Authentication Header ("AH") provides strong integrity, replay protection, and data authentication. AH protects as much of the IP datagram as it can. AH cannot protect fields that change nondeterministically between sender and receiver.

The Encapsulating Security Payload ("ESP") provides confidentiality over what it encapsulates, as well as the services that AH provides, but only over that which it encapsulates. ESP’s authentication services are optional, which allow ESP and AH to be used together on the same datagram without redundancy.

Two types of algorithms are used for IPsec, authentication and encryption algorithms. Authentication algorithms produce an integrity checksum value or "digest" based on the data and a key. The size of both the digest and the key are described in authentication algorithm pages. See, for example, authmd5h(7M) and authsha1(7M). Encryption algorithms encrypt data with a key. Encryption algorithms operate on data in units of a "block size". The size of both the block size and the key size are described in the encryption algorithm pages. See, for example, encrdes(7M) and encr3des(7M).

### Security Associations
Both AH and ESP use Security Associations (SAs), which are entities that specify security properties from one host to another. Two communicating machines need at least two SAs to communicate securely, unless they are using multicast, and then they can use the same multicast SA. SAs are managed through the pf_key(7P) interface. Automatic SA management is not yet available, but a command-line front-end is available by means of ipseckey(1M). An IPsec SA is identified by a tuple of <AH or ESP, destination IP address, and SPI>. The Security Parameters Index ("SPI") is an arbitrary 32-bit value that is transmitted on the wire with an AH or ESP packet. See ipsecah(7P) or ipsecesp(7P) for an explanation about where the SPI falls in a protected packet.

### Protection Policy and Enforcement Mechanisms
Mechanism and policy are separate. The policy for applying IPsec can be enforced in two places: on a system-wide level, or on a per-socket level. Configuring systemwide policy is done by the command ipsecconf(1M). Configuring per-socket policy will be discussed later in this section.

Systemwide IPsec policy is applied to incoming and outgoing datagrams. Some additional rules can be applied to outgoing datagrams because of the additional data known by the system. Inbound datagrams can either be accepted or dropped. The decision to drop or accept an inbound datagram is based on several criteria, which sometimes overlap or conflict. Conflict resolution is resolved by which rule is parsed.
first, with one exception. If a policy entry states that traffic should bypass all other policy, it will automatically be accepted. Outbound datagrams will either be sent with protection or without. If protection is applied, it can be either specific algorithms, or not. If policy normally would protect a datagram, it can be bypassed in either by an exception in systemwide policy, or by requesting a bypass in per-socket policy.

For intra-machine traffic, policies will be enforced, but actual security mechanisms will not be applied; rather, the outbound policy on an intra-machine packet will translate into an inbound packet that has had those mechanisms applied.

### Per-Socket Policy

The IPSEC_OPT socket option is used to set per-socket IPsec policy. The structure used for an IPSEC_OPT request is:

```c
typedef struct ipsec_req {
    uint_t ipsr_ah_req; /* AH request */
    uint_t ipsr_esp_req; /* ESP request */
    uint_t ipsr_self_encap_req; /* Self-Encap request */
    uint8_t ipsr_auth_alg; /* Auth algs for AH */
    uint8_t ipsr_esp_alg; /* Encr algs for ESP */
    uint8_t ipsr_esp_auth_alg; /* Auth algs for ESP */
} ipsec_req_t;
```

The IPsec request has fields for both AH and ESP. Algorithms can be specified, or not. The actual request for AH or ESP services can take one of the following values:

**IPSEC_PREF_NEVER** Bypass all policy. Only the superuser may request this service.

**IPSEC_PREF_REQUIRED** Regardless of other policy, require the use of the IPsec service.

The following value can be logically ORed to an IPSEC_PREF_REQUIRED value:

**IPSEC_PREF_UNIQUE** Regardless of other policy, enforce a unique SA for traffic originating from this socket.

The ipsec_self_encap_req is used to add an additional IP header outside the original one. This is in case IP options not normally encapsulated by ESP need to be. Algorithm values from `<net/pfkeyv2.h>` are as follows:

**SADB_AALG_MD5HMAC** This uses the MD5-HMAC (RFC 2403) algorithm for authentication. See authmd5h(7M).

**SADB_AALG_SHA1HMAC** This uses the SHA1-HMAC (RFC 2404) algorithm for authentication. See authsha1(7M).

**SADB_EALG_DESCBC** This uses the DES (RFC 2405) algorithm for encryption. See encrdes(7M).

**SADB_EALG_3DESCBC** This uses the Triple DES (RFC 2451) algorithm for encryption. See encr3des(7M).
An application should either use either the `getsockopt(3SOCKET)` or the `setsockopt(3SOCKET)` call to manipulate IPsec requests. For example:

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/pfkeyv2.h> /* For SADB_*ALG_* */
/* .... socket setup skipped */
rc = setsockopt(s, IPPROTO_IP, IP_SEC_OPT,
               (const char *)&ipsec_req, sizeof (ipsec_req_t));
```

While IPsec is an effective tool in securing network traffic, it will not make security problems disappear. Security issues beyond the mechanisms that IPsec offers may be discussed in a similar “Security Consideration” section within individual reference manual pages.

**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>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

ipsecconf(1M), ipseckey(1M), getsockopt(3SOCKET), setsockopt(3SOCKET), attributes(5), autenc3des(7M), inet(7P), ip(7P), ipsec(7P), ipsecconf(7P), ipsec(7P), ipsec(7P), pf_key(7P)


The ipsecah module ("AH") provides strong integrity, authentication, and partial sequence integrity (replay protection) to IP datagrams. AH protects the parts of the IP datagram that can be predicted by the sender as it will be received by the receiver. For example, the IP TTL field is not a predictable field, and is not protected by AH.

AH is inserted between the IP header and the transport header. The transport header can be TCP, UDP, ICMP, or another IP header, if tunnels are being used. See tun(7M).

AH is implemented as a module that is auto-pushed on top of IP. The entry /dev/ipsecah is used for tuning AH with ndd(1M), as well as to allow future authentication algorithms to be loaded on top of AH. Current authentication algorithms include HMAC-MD5 and HMAC-SHA-1. See authmd5h(7M) and authsha1(7P). Each authentication algorithm has its own key size and key format properties.

Without replay protection enabled, AH is vulnerable to replay attacks. AH does not protect against eavesdropping. Data protected with AH can still be seen by an adversary.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO

ipseconf(1M), ndd(1M), attributes(5), authmd5h(7M), authsha1(7P), ip(7P), ipsec(7P), ipsecesp(7P)

ipsecesp(7P)

NAME
ipsecesp, ESP – IPsec Encapsulating Security Payload

SYNOPSIS
drv/ipsecesp

DESCRIPTION
The ipsecesp module provides confidentiality, integrity, authentication, and partial sequence integrity (replay protection) to IP datagrams. The encapsulating security payload ("ESP") encapsulates its data, so it only protects the data that follows its beginning in the datagram. If the packet is a TCP packet, ESP will encapsulate the TCP header and its data only. If the packet is an IP in IP datagram, ESP will protect the inner IP datagram. Per-socket policy allows "self-encapsulation" so ESP can encapsulate IP options if it needs to. See ipsec(7P).

Unlike the authentication header ("AH") , ESP allows multiple kinds of datagram protection. To use a single form of datagram protection can expose vulnerabilities. For example, only ESP can be used to provide confidentiality. But protecting confidentiality alone exposes vulnerabilities in both replay attacks and cut-and-paste attacks. Similarly, if ESP protects only integrity and does not fully protect against eavesdropping, it may provide weaker protection than AH. See ipsecah(7P).

ESP is implemented as a module that is auto-pushed on top of IP. Use the /dev/ipsecesp entry to tune ESP with ndd(1M), as well as to allow future algorithms to be loaded on top of ESP. ESP allows encryption algorithms to be pushed on top of it, in addition to the authentication algorithms that can be used in AH. Authentication algorithms include HMAC-MD5 and HMAC-SHA-1. See authmd5h(7M) and authsha1(7P). Encryption algorithms include DES and Triple-DES. See encrdes(7M) and encr3des(7M). Each authentication and encryption algorithm has its key size and key format properties. Because of export laws in the United States, not all encryption algorithms will be available outside of the United States.

ESP without authentication exposes vulnerabilities to cut-and-paste cryptographic attacks, as well as eavesdropping attacks. When ESP is used without confidentiality, it is as vulnerable to replay as AH is.

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>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsrx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO
ipseconff(1M),ndd(1M),attributes(5),authmd5h(7M),authsha1(7P),encrdes(7M),
encr3des(7M),ip(7P),ipsec(7P),ipsecah(7P)

Due to United States export control laws, the encryption strength available on ESP will be weaker for versions of the SunOS sold outside the United States.
NAME

isdnio – ISDN interfaces

SYNOPSIS

```
#include <sun/audioio.h>
#include <sun/isdnio.h>

int ioctl(int fd, int command, /* arg */ ...);
```

DESCRIPTION

ISDN ioctl commands are a subset of ioctl(2) commands that perform a variety of control functions on Integrated Services Digital Network (ISDN) STREAMS devices. The arguments command and arg are passed to the file designated by fd and are interpreted by the ISDN device driver.

fd is an open file descriptor that refers to a stream. command determines the control function to be performed as described in the IOCTLS section of this document. arg represents additional information that is needed by command. The type of arg depends upon the command, but generally it is an integer or a pointer to a command-specific data structure.

Since these ISDN commands are a subset of ioctl and streamio(7I), they are subject to errors as described in those interface descriptions.

This set of generic ISDN ioctl commands is meant to control various types of ISDN STREAMS device drivers. The following paragraphs give some background on various types of ISDN hardware interfaces and data formats, and other device characteristics.

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.

Controllers, Interfaces, and Channels

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 ioctl’s 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 ioctl’s 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;
```

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
/* ISDN channels */
typedef enum {
    ISDN_CHAN_NONE = 0x0, /* No channel given */
    ISDN_CHAN_SELF, /* The channel performing the ioctl */
    ISDN_CHAN_HOST, /* Unix STREAM */
};
```
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
typedef enum {
    ISDN_TYPE_UNKNOWN = -1, /* Not known or applicable */
    ISDN_TYPE_SELF = 0, /* For queries, application may put this value into "type" to */
    /* ISDN interfaces */
    ISDN_CHAN_CTRL_MGT, /* Controller management */
    ISDN_CHAN_TE_MGT, /* Receives activation/deactivation */
    ISDN_CHAN_TE_D_TRACE, /* Trace device for protocol analysis apps */
    ISDN_CHAN_TE_D,
    ISDN_CHAN_TE_B1,
    ISDN_CHAN_TE_B2,
    ISDN_CHAN_NT_MGT, /* Receives activation/deactivation */
    ISDN_CHAN_NT_D_TRACE, /* Trace device for protocol analysis apps */
    ISDN_CHAN_NT_D,
    ISDN_CHAN_NT_B1,
    ISDN_CHAN_NT_B2,
    ISDN_CHAN_PRI_MGT,
    ISDN_CHAN_PRI_D,
    ISDN_CHAN_PRI_B0, ISDN_CHAN_PRI_B1,
    ISDN_CHAN_PRI_B2, ISDN_CHAN_PRI_B3,
    ISDN_CHAN_PRI_B4, ISDN_CHAN_PRI_B5,
    ISDN_CHAN_PRI_B6, ISDN_CHAN_PRI_B7,
    ISDN_CHAN_PRI_B8, ISDN_CHAN_PRI_B9,
    ISDN_CHAN_PRI_B10, ISDN_CHAN_PRI_B11,
    ISDN_CHAN_PRI_B12, ISDN_CHAN_PRI_B13,
    ISDN_CHAN_PRI_B14, ISDN_CHAN_PRI_B15,
    ISDN_CHAN_PRI_B16, ISDN_CHAN_PRI_B17,
    ISDN_CHAN_PRI_B18, ISDN_CHAN_PRI_B19,
    ISDN_CHAN_PRI_B20, ISDN_CHAN_PRI_B21,
    ISDN_CHAN_PRI_B22, ISDN_CHAN_PRI_B23,
    ISDN_CHAN_PRI_B24, ISDN_CHAN_PRI_B25,
    ISDN_CHAN_PRI_B26, ISDN_CHAN_PRI_B27,
    ISDN_CHAN_PRI_B28, ISDN_CHAN_PRI_B29,
    ISDN_CHAN_PRI_B30, ISDN_CHAN_PRI_B31,
    ISDN_CHAN_AUX0, ISDN_CHAN_AUX1, ISDN_CHAN_AUX2, ISDN_CHAN_AUX3,
    ISDN_CHAN_AUX4, ISDN_CHAN_AUX5, ISDN_CHAN_AUX6, ISDN_CHAN_AUX7
} isdn_chan_t;
```
isdnio(7I)

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` ioctl. 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;
```

```c
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;
```

Activation and Deactivation of ISDN Interfaces

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 ioctl**

- **ISDN_PH_ACTIVATE_REQ**
  
  Request ISDN physical layer activation. This command is valid for both TE and NT interfaces. `fd` must be a D-channel file descriptor. `arg` is ignored.

  TE activation will occur without use of the `ISDN_PH_ACTIVATE_REQ` ioctl if the device corresponding to the TE D-channel is open, “on”, and the ISDN switch is requesting activation.
ISDN_MPH_DEACTIVATE_REQ

*fd* must be an NT D-channel file descriptor. *arg* is ignored.

This command requests ISDN physical layer de-activation. This is not valid for TE interfaces. A TE 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

The *ISDN_INTERFACE_STATUS* ioctl retrieves the status and statistics of an ISDN interface. The requesting channel must own the interface whose status is being requested or the ioctl will fail. *fd* is the file descriptor for an ISDN interface management device. *arg* is a pointer to a struct *isdn_interface_info*. If the interface field is set to *ISDN_TYPE_SELF*, it will be changed in the returned structure to reflect the proper device-specific interface of the requesting *fd*.

```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 */
    unsigned int mph_di; /* Management: Deactivation Ind */
    unsigned int mph_ei1; /* Management: Error 1 Indication */
    unsigned int mph_ei2; /* Management: Error 2 Indication */
} isdn_interface_info_t;
```
unsigned int mph_ii_c; /* Management: Info Ind, connection */
unsigned int mph_ii_d; /* Management: Info Ind, disconn. */
} isdn_interface_info_t;

ISDN_CHANNEL_STATUS

The ISDN_CHANNEL_STATUS ioctl retrieves the status and statistics of an ISDN channel. The requesting channel must own the channel whose status is being requested or the ioctl will fail. fd is any file descriptor. arg is a pointer to a struct isdn_channel_info. If the interface field is set to ISDN_CHAN_SELF, it will be changed in the returned structure to reflect the proper device-specific channel of the requesting fd.

typedef struct isdn_channel_info {
    isdn_chan_t channel;
    enum isdn_iostate iostate;
    struct isdn_io_stats {
        ulong_t packets; /* packets transmitted or received */
        ulong_t octets; /* octets transmitted or received */
        ulong_t errors; /* errors packets transmitted or received */
    } transmit, receive;
} isdn_channel_info_t;

ISDN_PARAM_SET

fd is the file descriptor for a management device. arg is a pointer to a struct isdn_param. This command allows the setting of various ISDN physical layer parameters such as timers. This command uses the same arguments as the ISDN_PARAM_GET command.

ISDN_PARAM_GET

fd is the file descriptor for a management device. arg is a pointer to a struct isdn_param. This command provides for querying the value of a particular ISDN physical layer parameter.

typedef enum {
    ISDN_PARAM_NONE = 0,
    ISDN_PARAM_NT_T101, /* NT Timer, 5-30 s, in milliseconds */
    ISDN_PARAM_NT_T102, /* NT Timer, 25-100 ms, in milliseconds */
    ISDN_PARAM_TE_T103, /* TE Timer, 5-30 s, in milliseconds */
    ISDN_PARAM_TE_T104, /* TE Timer, 500-1000 ms, in milliseconds */
    ISDN_PARAM_MAINT, /* Manage the TE Maintenance Channel */
    ISDN_PARAM_ASG, /* Modify Activation State Machine Behavior */
    ISDN_PARAM_POWER, /* Take the interface online or offline */
    ISDN_PARAM_PAUSE, /* Paused if == 1, else not paused == 0 */
} isdn_param_tag_t;
enum isdn_param_asmb {
    ISDN_PARAM_TE_ASGB_CCIT88, /* 1988 bluebook */
    ISDN_PARAM_TE_ASGB_CTS2, /* Conformance Test Suite 2 */
};
typedef struct isdn_param {
    isdn_param_tag_t tag;
    union {
        unsigned int us; /* micro seconds */
    } isdn_param_tag_u;
} isdnio(7I)
unsigned int ms;       /* Timer value in ms */
unsigned int flag;     /* Boolean */
enum isdn_param_asmb asmb;
enum isdn_param_maint maint;
struct {
    isdn_chan_t channel;   /* Channel to Pause */
    int paused;            /* TRUE or FALSE */
} pause;
unsigned int reserved[2]; /* reserved, set to zero */
} value;
} isdn_param_t;

ISDN_PARAM_POWER
If an implementation provides power on and off functions, then power should be
on by default. If flag is ISDN_PARAM_POWER_OFF then a TE interface is forced
into state F0, NT interfaces are forced into state G0. If flag is
ISDN_PARAM_POWER_ON then a TE interface will immediately transition to state F3
when the TE D-channel is opened. If flag is one, an NT interface will transition to
state G1 when the NT D-channel is opened.

Implementations that do not provide ISDN_POWER return failure with errno set to
ENXIO. ISDN_POWER is different from ISDN_PH_ACTIVATE_REQ since CCITT
specification requires that if a BRI-TE interface device has power, then it permits
activation.

ISDN_PARAM_NT_T101
This parameter accesses the NT timer value T1. The CCITT recommendations
specify that timer T1 has a value from 5 to 30 seconds. Other standards may differ.

ISDN_PARAM_NT_T102
This parameter accesses the NT timer value T2. The CCITT recommendations
specify that timer T2 has a value from 25 to 100 milliseconds. Other standards may
differ.

ISDN_PARAM_TE_T103
This parameter accesses the TE timer value T3. The CCITT recommendations
specify that timer T3 has a value from 5 to 30 seconds. Other standards may differ.

ISDN_PARAM_TE_T104
This parameter accesses the TE timer value T4. The CTS2 specifies that timer T4 is
either not used or has a value from 500 to 1000 milliseconds. Other standards may
differ. CTS2 requires that timer T309 be implemented if T4 is not available.

ISDN_PARAM_MAINT
This parameter sets the multi-framing mode of a BRI-TE interface. For normal
operation this parameter should be set to ISDN_PARAM_MAINT_ECHO. Other uses
of this parameter are dependent on the definition and use of the BRI interface S and
Q channels.
ISDN_PARAM_ASMB

There are a few differences in the BRI-TE interface activation state machine standards. This parameter allows the selection of the appropriate standard. At this time, only ISDN_PARAM_TE_ASMB_CCITT88 and ISDN_PARAM_TE_ASMB_CTS2 are available.

ISDN_PARAM_PAUSE

This parameter allows a management device to pause the IO on a B-channel. pause.channel is set to indicate which channel is to be paused or un-paused. pause.paused is set to zero to un-pause and one to pause. fd is associated with an ISDN interface management device. arg is a pointer to a struct isdn_param.

ISDN_SET_LOOPBACK

fd is the file descriptor for an ISDN interface’s management device. arg is a pointer to an isdn_loopback_request_t structure.

typedef enum {
    ISDN_LOOPBACK_LOCAL,
    ISDN_LOOPBACK_REMOTE,
} isdn_loopback_type_t;

typedef enum {
    ISDN_LOOPBACK_B1 = 0x1,
    ISDN_LOOPBACK_B2 = 0x2,
    ISDN_LOOPBACK_D = 0x4,
    ISDN_LOOPBACK_E_ZERO = 0x8,
    ISDN_LOOPBACK_S = 0x10,
    ISDN_LOOPBACK_Q = 0x20,
} isdn_loopback_chan_t;

typedef struct isdn_loopback_request {
    isdn_loopback_type_t type;
    int channels;
} isdn_loopback_request_t;

An application can receive D-channel data during D-Channel loopback but cannot transmit data. The field type is the bitwise OR of at least one of the following values:

ISDN_LOOPBACK_B1 (0x1) /* loopback on B1-channel */
ISDN_LOOPBACK_B2 (0x2) /* loopback on B2-channel */
ISDN_LOOPBACK_D (0x4) /* loopback on D-channel */
ISDN_LOOPBACK_E_ZERO (0x8) /* force E-channel to Zero if */
* fd is for NT interface */
ISDN_LOOPBACK_S (0x10) /* loopback on S-channel */
ISDN_LOOPBACK_Q (0x20) /* loopback on Q-channel */

ISDN_RESET_LOOPBACK

arg is a pointer to an isdn_loopback_request_t structure.
ISDN_RESET_LOOPBACK turns off the selected loopback modes.

ISDN Data Format

The isdn_format_t type is meant to be a complete description of the various data modes and rates available on an ISDN interface. Several macros are available for setting the format fields. The isdn_format_t structure is shown below:
/* ISDN channel data format */
typedef enum {
    ISDN_MODE_NOTSPEC,    /* Not specified */
    ISDN_MODE_HDLC,      /* HDLC framing and error checking */
    ISDN_MODE_TRANSPARENT /* Transparent mode */
} isdn_mode_t;

/* Audio encoding types (from audioio.h) */
#define AUDIO_ENCODING_NONE (0) /* no encoding*/
#define AUDIO_ENCODING_ULAW (1) /* mu-law */
#define AUDIO_ENCODING_ALAW (2) /* A-law */
#define AUDIO_ENCODING_LINEAR (3) /* Linear PCM */
typedef struct isdn_format {
    isdn_mode_t mode;
    unsigned int sample_rate; /* sample frames/sec*/
    unsigned int channels; /* # interleaved chans */
    unsigned int precision; /* bits per sample */
    unsigned int encoding; /* data encoding */
} isdn_format_t;

/* These macros set the fields pointed */
/* to by the macro argument (isdn_format_t*)fp in preparation */
/* for the ISDN_SET_FORMAT ioctl. */

ISDN_SET_FORMAT_BRI_D(fp) /* BRI D-channel */
ISDN_SET_FORMAT_PRI_D(fp) /* PRI D-channel */
ISDN_SET_FORMAT_HDLC_B64(fp) /* BRI B-ch @ 56kbps */
ISDN_SET_FORMAT_HDLC_B56(fp) /* BRI B-ch @ 64kbps */
ISDN_SET_FORMAT_VOICE_ULAW(fp) /* BRI B-ch voice */
ISDN_SET_FORMAT_VOICE_ALAW(fp) /* BRI B-ch voice */
ISDN_SET_FORMAT_BRI_H(fp) /* BRI H-channel */

### ISDN Datapath Types

Every STREAMS stream that carries data to or from the ISDN serial interfaces is classified as a channel-stream datapath. A possible ISDN channel-stream datapath device name for a TE could be `/dev/isdn/0/te/b1`.

On some hardware implementations, it is possible to route the data from hardware channel to hardware channel completely within the chip or controller. This is classified as a channel-channel datapath. There does not need to be any open file descriptor for either channel in this configuration. Only when data enters the host and utilizes a STREAMS stream is this classified as an ISDN channel-stream datapath.

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 ioctl's describe operations on individual channels and the connection of multiple channels.

**ISDN_SET_FORMAT**

`fd` is a data channel, the management pseudo-channel associated with the data channel, or the management channel associated with the data channel’s interface or...
controller. arg is a pointer to a struct isdn_format_req. The ISDN_SET_FORMAT ioctl sets the format of an ISDN channel-stream datapath. It may be issued on both an open ISDN channel-stream datapath Stream or an ISDN Management Stream. Note that an open(2) call for a channel-stream datapath will fail if an ISDN_SET_FORMAT has never been issued after a reset, as the mode for all channel-stream datapaths is initially biased to ISDN_MODE_NOTSPEC. arg is a pointer to an ISDN format type (isdn_format_req_t).

typedef struct isdn_format_req {
    isdn_chan_t channel;
    isdn_format_t format;  /* data format */
    int reserved[4];  /* future use - must be 0 */
} isdn_format_req_t;

If there is not an open channel-stream datapath for a requested channel, the default format of that channel will be set for a subsequent open(2).

To modify the format of an open STREAM, the driver will disconnect the hardware channel, flush the internal hardware queues, set the new default configuration, and finally reconnect the data path using the newly specified format. Upon taking effect, all state information will be reset to initial conditions, as if a channel was just opened. It is suggested that the user flush the interface as well as consult the hardware specific documentation to insure data integrity.

If a user desires to connect more than one B channel, such as an H-channel, the B-channel with the smallest offset should be specified, then the precision should be specified multiples of 8. For an H-channel the precision value would be 16. The user should subsequently open the base B-channel. If any of the sequential B-channels are busy the open will fail, otherwise all of the B-channels that are to be used in conjunction will be marked as busy.

The returned failure codes and their descriptions are listed below:

EPERM  /* No permission for intended operation */
EINVAL  /* Invalid format request */
EIO  /* Set format attempt failed. */

ISDN_SET_CHANNEL
The ISDN_SET_CHANNEL ioctl sets up a data connection within an ISDN controller. The ISDN_SET_CHANNEL ioctl can only be issued from an ISDN management stream to establish or modify channel-channel datapaths. The ioctl parameter arg is a pointer to an ISDN connection request (isdn_conn_req_t*). Once a data path is established, data flow is started as soon as the path endpoints become active. Upon taking effect, all state information is reset to initial conditions, as if a channel was just opened.

The isdn_conn_req_t structure is shown below. The five fields include the receive and transmit ISDN channels, the number of directions of the data path, as well as the data format. The reserved field must always be set to zero.
/* Number of directions for data flow */
typedef enum {
    ISDN_PATH_NOCHANGE = 0,  /* Invalid value */
    ISDN_PATH_DISCONNECT,  /* Disconnect data path */
    ISDN_PATH_ONEWAY,  /* One way data path */
    ISDN_PATH_TWOWAY,  /* Bi-directional data path */
} isdn_path_t;

typedef struct isdn_conn_req {
    isdn_chan_t from;
    isdn_chan_t to;
    isdn_path_t dir;  /* uni/bi-directional or disconnect */
    isdn_format_t format;  /* data format */
    int reserved[4];  /* future use - must be 0 */
} isdn_conn_req_t;

To specify a read-only, write-only, or read-write path, or to disconnect a path, the dir
field should be set to ISDN_PATH_ONEWAY, ISDN_PATH_TWOWAY, and
ISDN_PATH_DISCONNECT respectively. To modify the format of a channel-channel
datapath, a user must disconnect the channel and then reconnect with the desired
format.

The returned failure codes and their descriptions are listed below:

EPERM  /* No permission for intended operation */
EBUSY  /* Connection in use */
EINVAL  /* Invalid connection request */
EIO  /* Connection attempt failed */

ISDN_GET_FORMAT
The ISDN_GET_FORMAT ioctl gets the ISDN data format of the channel-stream
datapath described by fd. arg is a pointer to an ISDN data format request type
(isdn_format_req_t *). ISDN_GET_FORMAT can be issued on any channel to
retrieve the format of any channel it owns. For example, if issued on the TE
management channel, the format of any other te channel can be retrieved.

ISDN_GETCONFIG
The ISDN_GETCONFIG ioctl is used to get the current connection status of all ISDN
channels associated with a particular management STREAM. ISDN_GETCONFIG
also retrieves a hardware identifier and the generic interface type. arg is an ISDN
connection table pointer (isdn_conn_tab_t *). The isdn_conn_tab_t structure
is shown below:

typedef struct isdn_conn_tab {
    char name[ISDN_ID_SIZE];  /* identification string */
    isdn_interface_t type;
    int maxpaths;  /* size in entries of app’s array int npaths; */
    /* number of valid entries returned by driver */
    isdn_conn_req_t *paths;  /* connection table in app’s memory */
} isdn_conn_tab_t;
The table contains a string which is the interface's unique identification string. The second element of this table contains the ISDN transmit and receive connections and configuration for all possible data paths for each type of ISDN controller hardware. Entries that are not connected will have a value of ISDN_NO_CHAN in the from and to fields. The number of entries will always be ISDN_MAX_CHANS, and can be referenced in the hardware specific implementation documentation. An isdn_conn_tab_t structure is allocated on a per controller basis.

SEE ALSO
getmsg(2), ioctl(2), open(2), poll(2), read(2), write(2), audio(7I), dbri(7D), streamio(7I)

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.

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.

EXAMPLE 1 SCSI Options

Create a file called /kernel/drv/isp.conf and add this line:

```plaintext
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)):

```plaintext
name="isp" parent="/iommu@f,e0000000/sbus@f,e0001000"
reg=1,0x10000,0x450

target1-scsei-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:

```plaintext
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
```
EXAMPLE 1 SCSI Options  (Continued)

Determine the register property values using the output of `prtconf(1M)` with the `-v` option:

QLGC,isp, instance #0

...  
Register Specifications:
  Bus Type=0x1, Address=0x10000, Size=450

EXAMPLE 2 ISP Properties

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

QLGC,isp, instance #2

Driver software properties:
  name <target0-TQ> length <0> -- <no value>.
  name <target0-wide> length <0> -- <no value>.
  name <target0-sync-speed> length <4>  
    value <0x000002f5>.
  name <scsi-options> length <4>  
    value <0x000003f8>.
  name <scsi-watchdog-tick> length <4>  
    value <0x0000000a>.
  name <scsi-tag-age-limit> length <4>  
    value <0x00000008>.
  name <scsi-reset-delay> length <4>  
    value <0x00000bb8>.

EXAMPLE 3 PCI Bus

To achieve the same setting of `SCSI-options` as in instance #0 above on a PCI machine, create a file called `/kernel/drv/isp.conf` and add the following entries.

```
name="isp" parent="/pci@1f,2000/pci@1"
  unit-address="4"  
  scsi-options=0x178  
  target3-scsi-options=0x58 scsi-initiator-id=6;
```

The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

To set `scsi-options` more specifically per device type, add the following line in the `/kernel/drv/isp.conf` file:
EXAMPLE 3 PCI Bus  (Continued)

device-type-scsi-options-list =
    "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;

All device which are of this specific disk type will have scsi-options set to 0x58.

scsi-options specified per target ID has the highest precedence, followed by
scsi-options per device type. Global (for all isp instances) scsi-options per
bus has the lowest precedence.

The system needs to be rebooted before the specified scsi-options take effect.

EXAMPLE 4 Driver Capabilities

The target driver needs to set capabilities in the isp driver in order to enable some
driver features. The target driver can query and modify these capabilities:
synchronous, tagged-qing, wide-xfer, auto-rqsense, qfull-retries, qfull-retry-interval. All other capabilities can only be queried.

By default, tagged-qing, auto-rqsense, and wide-xfer capabilities are disabled,
while disconnect, synchronous, and untagged-qing are enabled. These
capabilities can only have binary values (0 or 1). The default values for
qfull-retries and qfull-retry-interval are both 10. The qfull-retries
capability is a uchar_t (0 to 255) while qfull-retry-interval is a ushort_t (0
to 65535).

The target driver needs to enable tagged-qing and wide-xfer explicitly. The
untagged-qing capability is always enabled and its value cannot be modified,
because isp can queue commands even when tagged-qing is disabled.

Whenever there is a conflict between the value of scsi-options and a capability, the
value set in scsi-options prevails. Only whom != 0 is supported in the
scsi_ifsetcap(9F) call.

Refer to scsi_ifsetcap(9F) and scsi_ifgetcap(9F) for details.
SEE ALSO

prtconf(1M), driver.conf(4), attributes(5), scsi_abort(9F),
scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_reset(9F),
scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S),
scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

OpenBoot 3.x Command Reference Manual

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

QLogic Corporation, ISP1000 Firmware Interface Specification

QLogic Corporation, ISP1020 Firmware Interface Specification

QLogic Corporation, ISP1000 Technical Manual

QLogic Corporation, ISP1020a/1040a Technical Manual

QLogic Corporation, Differences between the ISP1020a/1040a and the ISP1020B/1040B - Application Note

DIAGNOSTICS

The messages described below may appear on the system console as well as being logged.

The first set of messages may be displayed while the isp driver is first trying to attach. All of these messages mean that the isp driver was unable to attach. These messages are preceded by "isp<number>", where "<number>" is the instance number of the ISP Host Bus Adapter.

Device in slave-only slot, unused
   The SBus device has been placed in a slave-only slot and will not be accessible;
   move to non-slave-only SBus slot.

Device is using a hilevel intr, unused
   The device was configured with an interrupt level that cannot be used with this
   isp driver. Check the device.

Failed to alloc soft state
   Driver was unable to allocate space for the internal state structure. Driver did not
   attach to device; SCSI devices will be inaccessible.

Bad soft state
   Driver requested an invalid internal state structure. Driver did not attach to device;
   SCSI devices will be inaccessible.

Unable to map registers
   Driver was unable to map device registers; check for bad hardware. Driver did not
   attach to device; SCSI devices will be inaccessible.
Cannot add intr
  Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device; SCSI devices will be inaccessible.

Unable to attach
  Driver was unable to attach to the hardware for some reason that may be printed. Driver did not attach to device; SCSI devices will be inaccessible.

The next set of messages can be displayed at any time. They will be printed with the full device pathname followed by the shorter form described above.

Firmware should be < 0x<number> bytes
  Firmware size exceeded allocated space and will not download firmware. This could mean that the firmware was corrupted somehow. Check the isp driver.

Firmware checksum incorrect
  Firmware has an invalid checksum and will not be downloaded.

Chip reset timeout
  ISP chip failed to reset in the time allocated; may be bad hardware.

Stop firmware failed
  Stopping the firmware failed; may be bad hardware.

Load ram failed
  Unable to download new firmware into the ISP chip.

DMA setup failed
  The DMA setup failed in the host adapter driver on a scsi_pkt. This will return TRAN_BADPKT to a SCSA target driver.

Bad request pkt
  The ISP Firmware rejected the packet as being set up incorrectly. This will cause the isp driver to call the target completion routine with the reason of CMD_TRAN_ERR set in the scsi_pkt. Check the target driver for correctly setting up the packet.

Bad request pkt header
  The ISP Firmware rejected the packet as being set up incorrectly. This will cause the isp driver to call the target completion routine with the reason of CMD_TRAN_ERR set in the scsi_pkt. Check the target driver for correctly setting up the packet.

Polled command timeout on <number>.<number>
  A polled command experienced a timeout. The target device, as noted by the target lun (<number>.<number>) information, may not be responding correctly to the command, or the ISP chip may be hung. This will cause an error recovery to be initiated in the isp driver. This could mean a bad device or cabling.

SCSI Cable/Connection problem

Hardware/Firmware error
  The ISP chip encountered a firmware error of some kind. The problem is probably due to a faulty scsi cable or improper cable connection. This error will cause the isp driver to do error recovery by resetting the chip.
Received unexpected SCSI Reset

The ISP chip received an unexpected SCSI Reset and has initiated its own internal
error recovery, which will return all the scsi_pkt with reason set to CMD_RESET.

Fatal timeout on target <number>.<number>

The isp driver found a command that had not completed in the correct amount of
time; this will cause error recovery by the isp driver. The device that experienced
the timeout was at target lun (<number>.<number>).

Fatal error, resetting interface

This is an indication that the isp driver is doing error recovery. This will cause all
outstanding commands that have been transported to the isp driver to be
completed via the scsi_pkt completion routine in the target driver with reason of
CMD_RESET and status of STAT_BUS_RESET set in the scsi_pkt.
The \texttt{kb} STREAMS module processes byte streams generated by a keyboard attached to a CPU serial port. Definitions for altering keyboard translation and reading events from the keyboard are contained in \texttt{<sys/kbio.h>} and \texttt{<sys/kbd.h>}. The \texttt{kb} STREAMS module utilizes a set of keyboard tables to recognize which keys have been typed. Each translation table is an array of 128 16-bit words (unsigned shorts). If a table entry is less than 0x100, the entry is treated as an ISO 8859/1 character. Higher values indicate special characters that invoke more complicated actions.

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**: \texttt{firm_events} are reported.
- **TR_UNTRANS_EVENT**: \texttt{firm_events} containing unencoded keystation codes are reported for all input events within the window system.

All instances of the \texttt{kb} module share seven translation tables that convert raw keystation codes to event values. The tables are:

- **Unshifted**: Used when a key is depressed and no shifts are in effect.
- **Shifted**: Used when a key is depressed and a Shift key is held down.
- **Caps Lock**: Used when a key is depressed and Caps Lock is in effect.
- **Alt Graph**: Used when a key is depressed and the Alt Graph key is held down.
- **Num Lock**: Used when a key is depressed and Num Lock is in effect.
Controlled Used when a key is depressed and the Control key is held down.
(Regardless of whether a Shift key or the Alt Graph is being held down, or whether Caps Lock or Num Lock is in effect).

Key Up Used when a key is released.

Each key on the keyboard has a keystation code that represents a number from 0 to 127. The number is used as an index into the translation table that is currently in effect. If the corresponding entry in the translation table is a value from 0 to 255, the value is treated as an ISO 8859/1 character, and the character is the result of the translation.

If the entry in the translation table is higher than 255, it is a special entry. Special entry values are classified according to the value of the high-order bits. The high-order value for each class is defined as a constant, as shown below. When added to the constant, the value of the low-order bits distinguish between keys within each class:

SHIFTKEYS 0x100
A shift key. The value of the particular shift key is added to determine which shift mask to apply:

- CAPSLOCK 0 Caps Lock key.
- SHIFTLOCK 1 “Shift Lock” key.
- LEFTSHIFT 2 Left-hand Shift key.
- RIGHTSHIFT 3 Right-hand Shift key.
- LEFTCTRL 4 Left-hand (or only) Control key.
- RIGHTCTRL 5 Right-hand Control key.
- ALTGRAPH 9 Alt Graph key.
- ALT 10 Alternate or Alt key.
- NUMLOCK 11 Num Lock key.

BUCKYBITS 0x200
Used to toggle mode-key-up/down status without altering the value of an accompanying ISO 8859/1 character. The actual bit-position value, minus 7, is added.

- METABIT 0 The Meta key was pressed along with the key. This is the only user-accessible bucky bit. It is ORed in as the 0x80 bit; since this bit is a legitimate bit in a character, the only way to distinguish between, for example, 0xA0 as META+0x20 and 0xA0 as an 8-bit character is to watch for META key up and META key down events and keep track of whether the META key was down.

- SYSTEMBIT 1 The System key was pressed. This is a place holder to indicate which key is the system-abort key.
FUNNY 0x300
Performs various functions depending on the value of the low 4 bits:

NOP 0x300  Does nothing.
OOPS 0x301  Exists, but is undefined.
HOLE 0x302  There is no key in this position on the keyboard, and the position-code should not be used.
RESET 0x306  Keyboard reset.
ERROR 0x307  The keyboard driver detected an internal error.
IDLE 0x308  The keyboard is idle (no keys down).
COMPOSE 0x309  The COMPOSE key; the next two keys should comprise a two-character COMPOSE key sequence.
NONL 0x30A  Used only in the Num Lock table; indicates that this key is not affected by the Num Lock state, so that the translation table to use to translate this key should be the one that would have been used had Num Lock not been in effect.

0x30B — 0x30F  Reserved for non-parameterized functions.

FA_CLASS 0x400
A floating accent or “dead key.” When this key is pressed, the next key generates an event for an accented character; for example, “floating accent grave” followed by the “a” key generates an event with the ISO 8859/1 code for the “a with grave accent” character. The low-order bits indicate which accent; the codes for the individual “floating accents” are as follows:

FA_UMLAUT 0x400  umlaut
FA_CFLEX 0x401  circumflex
FA_TILDE 0x402  tilda
FA_CEDILLA 0x403  cedilla
FA_ACUTE 0x404  acute accent
FA_GRAVE 0x405  grave accent

STRING 0x500
The low-order bits index a table of strings. When a key with a STRING entry is depressed, the characters in the null-terminated string for that key are sent, character-by-character. The maximum length is defined as:

KTAB_STRLEN 10  Individual string numbers are defined as:

HOMEARROW 0x00
String numbers 0x05 — 0x0F are available for custom entries.

**FUNCKEYS 0x600**
There are 64 keys reserved for function keys. The actual positions are usually on the left/right/top/bottom of the keyboard.

The next-to-lowest 4 bits indicate the group of function keys:

- LEFTFUNC 0x600
- RIGHTFUNC 0x610
- TOPFUNC 0x610
- 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) \( (\text{BOTTOMFUNC}+(n)-1) \)

**PADKEYS 0x700**
A "numeric keypad key." These entries should appear only in the Num Lock translation table; when Num Lock is in effect, these events will be generated by pressing keys on the right-hand keypad. The low-order bits indicate which key. The codes for the individual keys are:

- PADEQUAL 0x700 \"=" key
- PADSLASH 0x701 \"/" key
- PADSTAR 0x702 \"*\" key
- PADMINUS 0x703 \"-\" key
- PADSEP 0x704 \"\," key
- PAD7 0x705 \"7\" key
- PAD8 0x706 \"8\" key
When a function key is pressed in TR.ASCII mode, the following escape sequence is sent:

```
ESC[0 . . . . 9z
```

where ESC is a single escape character and “0 . . . 9” indicates the decimal representation of the function-key value. For example, function key R1 sends the sequence:

```
ESC[208z
```

because the decimal value of RF(1) is 208. In TR.EVENT mode, if there is a VUID event code for the function key in question, an event with that event code is generated; otherwise, individual events for the characters of the escape sequence are generated.

When started, the kb STREAMS module is in the compatibility mode. When the keyboard is in the TR.EVENT translation mode, ISO 8859/1 characters from the upper half of the character set (that is, characters with the eighth bit set), are presented as events with codes in the ISO_FIRST range (as defined in <<sys/vuid_event.h>>). For backwards compatibility with older versions of the keyboard driver, the event code is ISO_FIRST plus the character value. When compatibility mode is turned off, ISO 8859/1 characters are presented as events with codes equal to the character code.

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

- **KIOCTRANS**: Pointer to an int. The translation mode is set to the value in the int pointed to by the argument.
- **KIOCGTRANS**: Pointer to an int. The current translation mode is stored in the int pointed to by the argument.
ioctl() requests for changing and retrieving entries from the keyboard translation table use the kiockeymap structure:

```c
struct kiockeymap {
    int kio_tablemask; /* Translation table (one of: 0, CAPSMASK,
                        * SHIFTMASK, CTLRMASK, UPMASK,
                        * ALTGRAPHMASK, NUMLOCKMASK)
    #define KIOCABORT1 -1 /* Special "mask": abort1 keystation */
    #define KIOCABORT2 -2 /* Special "mask": abort2 keystation */
    uchar_t kio_station; /* Physical keyboard key station (0-127) */
    ushort_t kio_entry; /* Translation table station's entry */
    char kio_string[10]; /* Value for STRING entries (null terminated) */
};
```

KIOCSKEY Pointer to a kiockeymap structure. The translation table entry referred to by the values in that structure is changed. The kio_tablemask request specifies which of the following translation tables contains the entry to be modified:

- **UPMASK 0x0080** “Key Up” translation table.
- **NUMLOCKMASK 0x0800** “Num Lock” translation table.
- **CTRLMASK 0x0030** “Controlled” translation table.
- **ALTGRAPHMASK 0x0200** “Alt Graph” translation table.
- **SHIFTMASK 0x000E** “Shifted” translation table.
- **CAPSMASK 0x0001** “Caps Lock” translation table.
- **(No shift keys pressed or locked)** “Unshifted” translation table.

The kio_station request specifies the keystation code for the entry to be modified. The value of kio_entry is stored in the entry in question. If kio_entry is between STRING and STRING+15, the string contained in kio_string is copied to the appropriate string table entry. This call may return EINVAL if there are invalid arguments.

Special values of kio_tablemask can affect the two step “break to the PROM monitor” sequence. The usual sequence is Li-a or Stop. If kio_tablemask is KIOCABORT1, then the value of kio_station is set to be the first keystation in the sequence. If kio_tablemask is KIOCABORT2 then the value of kio_station is set
to be the second keystation in the sequence. An attempt to change the "break to the
PROM monitor" sequence without having superuser permission results in an EPERM
error.

**KIOCGKEY**

The argument is a pointer to a kiockeymap structure. The current
value of the keyboard translation table entry specified by
kio_tablemask and kio_station is stored in the structure
pointed to by the argument. This call may return EINVAL if there
are invalid arguments.

**KIOCTYPE**

The argument is a pointer to an int. A code indicating the type of
the keyboard is stored in the int pointed to by the argument:

- **KB_SUN3**  Sun Type 3 keyboard
- **KB_SUN4**  Sun Type 4 keyboard
- **KB_ASCII** ASCII terminal masquerading as keyboard
- **KB_PC**    Type 101 PC keyboard
- **KB_DEFAULT** Stored in the int pointed to by the argument if
  the keyboard type is unknown. In case of error,
  -1 is stored in the int pointed to by the argument.

**KIOCLAYOUT**

The argument is a pointer to an int. On a Sun Type 4 keyboard,
the layout code specified by the keyboard’s DIP switches is stored
in the int pointed to by the argument.

**KIOCCMD**

The argument is a pointer to an int. The command specified by
the value of the int pointed to by the argument is sent to the
keyboard. The commands that can be sent are:

- **KBD_CMD_RESET**  Reset keyboard as if power-up.
- **KBD_CMD_BELL**   Turn on the bell.
- **KBD_CMD_NOBELL** Turn off the bell.
- **KBD_CMD_CLICK**  Turn on the click annunciator.
- **KBD_CMD_NOCLICK** Turn off the click annunciator.

Commands to the Sun Type 4 keyboard:

- **KBD_CMD_SETLED** Set keyboard LEDs.
- **KBD_CMD_GETLAYOUT** Request that keyboard indicate
  layout.

Inappropriate commands for particular keyboard types are ignored. Since there is no
reliable way to get the state of the bell or click (because the keyboard cannot be

---

292  man pages section 7: Device and Network Interfaces  •  Last Revised 14 May 1999
queried and a process could do writes to the appropriate serial driver —
circumventing this `ioctl()` request — an equivalent `ioctl()` to query its state is not
provided.

**KIOCSLED**

The argument is a pointer to an `char`. On the Sun Type 4
keyboard, the LEDs are set to the value specified in that `char`. The
values for the four LEDs are:

- **LED_CAPS_LOCK**
  “Caps Lock” light.
- **LED_COMPOSE**
  “Compose” light.
- **LED_SCROLL_LOCK**
  “Scroll Lock” light.
- **LED_NUM_LOCK**
  “Num Lock” light.

On some Japanese layouts, the value for the fifth LED is:

- **LED_KANA**
  “Kana” light.

**KIOCGLED**

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

**KIOCSCOMPAT**

Pointer to an `int`. “Compatibility mode” is turned on if the `int`
has a value of 1, and is turned off if the `int` has a value of 0.

**KIOCGCOMPAT**

Pointer to an `int`. The current state of “compatibility mode” is
stored in the `int` pointed to by the argument.

The following `ioctl()` request allows the default effect of the keyboard abort
sequence to be changed.

**KIOCSKABORTEN**

Pointer to an `int`. The keyboard abort sequence effect (typically L1-A or Stop-A on
the keyboard on SPARC systems, F1-A on IA systems, and BREAK on the serial
console device) is enabled if the `int` has a value of KIOCABORTENABLE(1). If the
value is KIOCABORTDISABLE(0), the keyboard abort sequence effect is disabled.
If the value is KIOCABORTALTERNATE(2), the Alternate Break sequence is in
effect and is defined by the serial console drivers `zs(7D)` `se(7D)` and `asy(7D)`. Any
other value of the parameter for this `ioctl()` is treated as enable. The Alternate
Break sequence is applicable to the serial console devices only. When the Alternate
Break sequence is in effect, binary protocols including PPP, SLIP, file transfer and
others should not be run over the console serial port.

This `ioctl()` will be active and retain state even if there is no physical keyboard in
the system. The default effect (enable) causes the operating system to suspend and
enter the kernel debugger (if present) or the system prom (on most systems with
OpenBoot proms). The default effect is enabled on most systems, but may be
different on server systems with key switches in the ‘secure’ position. On these
systems, the effect is always disabled when the key switch is in the ‘secure’
position. This `ioctl()` returns EPERM if the caller is not the superuser.

---

kb(7M)
These ioctl() requests are supported for compatibility with the system keyboard device /dev/kbd.

**KIOCSDIRECT**
Has no effect.

**KIOCGDIRECT**
Always returns 1.

**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>Stable</td>
</tr>
</tbody>
</table>

**SEE ALSO**
kbd(1), loadkeys(1), kadb(1M), keytables(4), attributes(5), zs(7D), se(7D), asy(7D), termio(7I)

**NOTES**
Many of the keyboards released after Sun Type 4 keyboard also report themselves as Sun Type 4 keyboard.
kdmouse(7D)

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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5), vuidmice(7M)

### NAME
kstat – kernel statistics driver

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

### FILES
/dev/kstat kernel statistics driver

### SEE ALSO
kstat(3KSTAT), kstat(9S)
ksyms(7D)

NAME     ksym – kernel symbols

SYNOPSIS /dev/ksyms

DESCRIPTION The file /dev/ksyms is a character special file that allows read-only access to an ELF format image containing two sections: a symbol table and a corresponding string table. The contents of the symbol table reflect the symbol state of the currently running kernel. You can determine the size of the image with the fstat() system call. The recommended method for accessing the /dev/ksyms file is by using the ELF access library. See elf(3ELF) for details. If you are not familiar with ELF format, see a.out(4).

/dev/ksyms is an executable for the processor on which you are accessing it. It contains ELF program headers which describe the text and data segment(s) in kernel memory. Since /dev/ksyms has no text or data, the fields specific to file attributes are initialized to NULL. The remaining fields describe the text or data segment(s) in kernel memory.

Symbol table

The SYMTAB section contains the symbol table entries present in the currently running kernel. This section is ordered as defined by the ELF definition with locally-defined symbols first, followed by globally-defined symbols. Within symbol type, the symbols are ordered by kernel module load time. For example, the kernel file symbols are first, followed by the first module’s symbols, and so on, ending with the symbols from the last module loaded.

The section header index (st_shndx) field of each symbol entry in the symbol table is set to SHN_ABS, because any necessary symbol relocations are performed by the kernel link editor at module load time.

String table

The STRTAB section contains the symbol name strings that the symbol table entries reference.

SEE ALSO kernel(1M), stat(2), elf(3ELF), kvm_open(3KVM), a.out(4), mem(7D)

WARNINGS The kernel is dynamically configured. It loads kernel modules when necessary. Because of this aspect of the system, the symbol information present in the running system can vary from time to time, as kernel modules are loaded and unloaded.

When you open the /dev/ksyms file, you have access to an ELF image which represents a snapshot of the state of the kernel symbol information at that instant in time. While the /dev/ksyms file remains open, kernel module autounloading is disabled, so that you are protected from the possibility of acquiring stale symbol data. Note that new modules can still be loaded, however. If kernel modules are loaded while you have the /dev/ksyms file open, the snapshot held by you will not be updated. In order to have access to the symbol information of the newly loaded modules, you must close and reopen the /dev/ksyms file.
modules, you must first close and then reopen the /dev/ksyms file. Be aware that the size of the /dev/ksyms file will have changed. You will need to use the fstat() function (see stat(2)) to determine the new size of the file.

Avoid keeping the /dev/ksyms file open for extended periods of time, either by using kvm_open(3KVM) of the default namelist file or with a direct open. There are two reasons why you should not hold /dev/ksyms open. First, the system's ability to dynamically configure itself is partially disabled by the locking down of loaded modules. Second, the snapshot of symbol information held by you will not reflect the symbol information of modules loaded after your initial open of /dev/ksyms.

Note that the ksym 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 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 driver to be loaded (if necessary). You can then use the file descriptor from this open in a fstat() system call to get the file’s size.

NOTES
The kernel virtual memory access library (libkvm) routines use /dev/ksyms as the default namelist file. See kvm_open(3KVM) for details.
The ldterm STREAMS module provides most of the termio(7I) terminal interface. The vis module does not perform the low-level device control functions specified by flags in the c_cflag word of the termio/termios structure, or by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the c_iflag word of the termio/termios structure. Those functions must be performed by the driver or by modules pushed below the ldterm module. The ldterm module performs all other termio/termios functions, though some may require the cooperation of the driver or modules pushed below ldterm and may not be performed in some cases. These include the IXOFF flag in the c_iflag word and the delays specified in the c_oflag word.

The ldterm module also handles single and multi-byte characters from various codesets including both Extended Unix Code (EUC) and non-EUC codesets.

The remainder of this section describes the processing of various STREAMS messages on the read- and write-side.

### Read-side Behavior

Various types of STREAMS messages are processed as follows:

- **M_BREAK**: Depending on the state of the BRKINT flag, either an interrupt signal is generated or the message is treated as if it were an M_DATA message containing a single ASCII NUL character when this message is received.

- **M_DATA**: This message is normally processed using the standard termio input processing. If the ICANON flag is set, a single input record ("line") is accumulated in an internal buffer and sent upstream when a line-terminating character is received. If the ICANON flag is not set, other input processing is performed and the processed data are passed upstream.

  If output is to be stopped or started as a result of the arrival of characters (usually CNTRL-Q and CNTRL-S), M_STOP and M_START messages are sent downstream. If the IXOFF flag is set and input is to be stopped or started as a result of flow-control considerations, M_STOPI and M_STARTI messages are sent downstream.

  M_DATA messages are sent downstream, as necessary, to perform echoing.

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

All other messages are passed upstream unchanged.

Various types of STREAMS messages are processed as follows:

- **M_FLUSH**
  The write queue of the module is flushed of all its data messages and the message is passed downstream.

- **M_IOCTL**
  The function of this ioctl is performed and the message is passed downstream in most cases. The TCFLSH and TCXONC ioctls can be performed entirely in the ldterm module, so the reply is sent upstream and the message is not passed downstream.

- **M_DATA**
  If the OPOST flag is set, or both the XCASE and ICANON flags are set, output processing is performed and the processed message is passed downstream along with any M_DELAY messages generated. Otherwise, the message is passed downstream without change.

- **M_CTL**
  If the size of the data buffer associated with the message is the size of struct iocblk, ldterm will perform functional negotiation to determine where the termio(7I) processing is to be done. If the command field of the iocblk structure (ioc_cmd) is set to MC_NO_CANON, the input canonical processing normally performed on M_DATA messages is disabled and those messages are passed upstream unmodified. (This is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in TIOCREMOTE mode connected to a program that performs this processing). If the command is MC_DO_CANON, all input processing is enabled. If the command is MC_PART_CANON, then an M_DATA message containing a termios structure is expected to be attached to the original M_CTL message. The ldterm module will examine the iflag, oflag, and lflag fields of the termios structure and from that point on, will process only those flags that have not been turned ON. If none of the above commands are found, the message is ignored. In any case, the message is passed upstream.

- **M_FLUSH**
  The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.

- **M_IOCTL**
  The data contained within the message, which is to be returned to the process, are augmented if necessary, and the message is passed upstream.

All other messages are passed downstream unchanged.

**IOCTLS**

The ldterm module processes the following TRANSPARENT ioctls. All others are passed downstream.
TCGETS/TCGETA
The message is passed downstream. If an acknowledgment is seen, the data provided by the driver and modules downstream are augmented and the acknowledgement is passed upstream.

TCSETS/TCSETSW/TCSETSF/TCSETA/TCSETAW/TCSETAF
The parameters that control the behavior of the ldterm module are changed. If a mode change requires options at the stream head to be changed, an M_SETOPTS message is sent upstream. If the ICANON flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If the TOSTOP flag is turned on or off, the tostop mode at the stream head is turned on or off, respectively. In any case, ldterm passes the ioctl on downstream for possible additional processing.

TCFLSH
If the argument is 0, an M_FLUSH message with a flag byte of FLUSHR is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHW is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHRW is sent downstream and placed on the read queue.

TCXONC
If the argument is 0 and output is not already stopped, an M_STOP message is sent downstream. If the argument is 1 and output is stopped, an M_START message is sent downstream. If the argument is 2 and input is not already stopped, an M_STOPI message is sent downstream. If the argument is 3 and input is stopped, an M_STARTI message is sent downstream.

TCSBRK
The message is passed downstream, so the driver has a chance to drain the data and then send an M_IOCACK message upstream.

EUC_WSET
This call takes a pointer to an eucioc structure, and uses it to set the EUC line discipline’s local definition for the code set widths to be used for subsequent operations. Within the stream, the line discipline may optionally notify other modules of this setting using M_CTL messages. When this call is received and the euciocstructure contains valid data, the line discipline changes into EUC handling mode once the euciocdata is completely transferred to an internal data structure.

EUC_WGET
This call takes a pointer to an eucioc structure, and returns in it the EUC code set widths currently in use by the EUC line discipline. If the current codeset of the line discipline is not an EUC one, the result is meaningless.

SEE ALSO
termios(3C), console(7D), termio(7I)
STREAMS Programming Guide
le(7D)

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.

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.

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 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 (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, 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)

`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.

```bash
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 following line to the end of the file:

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

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 MAC type is DL_CSMACD or DL_TPR as determined by the driver linked under llc1. If the driver reports that it is DL_ETHER, it will be changed to DL_CSMACD; otherwise the type is the same as the MAC type.
- The sap length value is -1, meaning the physical address component is followed immediately by a 1-octet sap component within the DLSAP address.
- The service mode is DL_CLDLS.
- The MAC type is DL_CSMACD or DL_TPR as determined by the driver linked under llc1. If the driver reports that it is DL_ETHER, it will be changed to DL_CSMACD; otherwise the type is the same as the MAC type.
- The dlsap address length is 7.
- No optional quality of service (QOS) support is included at present, so the QOS fields should be initialized to 0.
The DLPI version is \texttt{DL\_VERSION\_2}.

- The provider style is \texttt{DL\_STYLE2}.
- The broadcast address value is the broadcast address returned from the lower level driver.

Once in the \texttt{DL\_ATTACHED} state, the user must send a \texttt{DL\_BIND\_REQ} to associate a particular Service Access Point (SAP) with the stream. The \texttt{llc1} driver interprets the \texttt{sap} field within the \texttt{DL\_BIND\_REQ} as an IEEE 802.2 “SAP,” therefore valid values for the \texttt{sap} field are in the \([0-0xFF]\) range with only even values being legal.

The \texttt{llc1} driver DLSAP address format consists of the 6-octet physical (e.g., Ethernet) address component followed immediately by the 1-octet \texttt{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 \texttt{DL\_INFO\_ACK} primitive to compose and decompose DLSAP addresses. The \texttt{sap} length, full DLSAP length, and \texttt{sap}/physical ordering are included within the \texttt{DL\_INFO\_ACK}. The physical address length can be computed by subtracting the absolute value of the \texttt{sap} length from the full DLSAP address length or by issuing the \texttt{DL\_PHYS\_ADDR\_REQ} to obtain the current physical address associated with the stream.

Once in the \texttt{DL\_BOUND} state, the user may transmit frames on the LAN by sending \texttt{DL\_UNITDATA\_REQ} messages to the \texttt{llc1} driver. The \texttt{llc1} driver will route received frames up all open and bound streams having a \texttt{sap} which matches the IEEE 802.2 DSAP as \texttt{DL\_UNITDATA\_IND} messages. Received frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the \texttt{DL\_UNITDATA\_REQ} and \texttt{DL\_UNITDATA\_IND} messages consists of both the \texttt{sap} (type) and physical (Ethernet) components.

In addition to the mandatory, connectionless DLPI message set, the driver additionally supports the following primitives:

- The \texttt{DL\_ENABMULTI\_REQ} and \texttt{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 \texttt{ppa}.

- The \texttt{DL\_PHYS\_ADDR\_REQ} primitive returns the 6-octet physical address currently associated (attached) to the stream in the \texttt{DL\_PHYS\_ADDR\_ACK} primitive. This primitive is valid only in states following a successful \texttt{DL\_ATTACH\_REQ}.

- The \texttt{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>IA</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), dlpi(7P)
NAME | llc2 – Class II logical link control driver

DESCRIPTION | The llc2 logical link control driver interfaces network software (NetBIOS, SNA, OSI, etc.) running under the Solaris operating environment to a physical LAN network controlled by one of the supported communications adapters. The llc2 driver, which appears as a STREAMS driver to the network software, resides in the kernel and is accessed by standard UNIX STREAMS functions.

This version of the llc2 driver includes support for both connectionless and connection-oriented logical link control class II (llc2) operations for Ethernet, Token Ring, and FDDI adapters when accessed through the appropriate Solaris MAC layer driver. The Data Link Provider Interface (DLPI) to the llc2 driver enables multiple and different protocol stacks, (including NetBIOS and SNA), to operate simultaneously over one or more local area networks.

To start the llc2 driver by default, rename file /etc/llc2/llc2_start.default to /etc/llc2/llc2_start. This allows the /etc/rc2.d/S40llc2 script to build up the configuration file for each ppa interface in /etc/llc2/default/llc2.* and start llc2 on each interface. To verify the configuration files, manually run /usr/lib/llc2/llc2_autoconfig.

For more information on the llc2 driver, see the IEEE standard 802.2 Logical Link Control.

OBTAINING LLC2 STATISTICS | You can obtain LLC2 statistics or reset the statistics counter to zero using the ILD_LLC2 ioctl. The ILD_LLC2 ioctl has a number of subcommands. The following retrieve LLC2 statistics:

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC2_GET_STA_STATS</td>
<td>Get station statistics</td>
</tr>
<tr>
<td>LLC2_GET_SAP_STATS</td>
<td>Get SAP statistics</td>
</tr>
<tr>
<td>LLC2_GET_CON_STATS</td>
<td>Get connection statistics</td>
</tr>
</tbody>
</table>

The structure used depends on the subcommand sent.

LLC2_GET_STA_STATS | The LLC2_GET_STA_STATS command retrieves statistics on a particular Physical Point of Attachment (PPA).

When sending the LLC2_GET_STA_STATS command, the llc2GetStaStats structure is used:

```c
typedef struct llc2GetStaStats {
    uint_t ppa;
    uint_t cmd;
    uchar_t clearFlag;
    uchar_t state;
    ushort_t numSaps;
};
```

Device and Network Interfaces 309
The members of the structure are:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>LLC2_GET_STA_STATS</td>
</tr>
<tr>
<td>clearFlag</td>
<td>Clear counters flag. Set this to 0 to retrieve statistics and to 1 to reset all counters to 0.</td>
</tr>
<tr>
<td>state</td>
<td>Station component state. Possible values are ?????</td>
</tr>
<tr>
<td>numSaps</td>
<td>Number of active SAPs in the saps array</td>
</tr>
<tr>
<td>saps</td>
<td>An array of active SAP values</td>
</tr>
<tr>
<td>nullSapXidCmdRcvd</td>
<td>Number of NULL SAP XID commands received</td>
</tr>
<tr>
<td>nullSapXidRspSent</td>
<td>Number of NULL SAP XID responses sent</td>
</tr>
<tr>
<td>nullSapTestCmdRcvd</td>
<td>Number of NULL SAP TEST commands received</td>
</tr>
<tr>
<td>nullSapTestRspSent</td>
<td>Number of NULL SAP TEST responses sent</td>
</tr>
<tr>
<td>outOfState</td>
<td>Number of invalid events received</td>
</tr>
<tr>
<td>allocFail</td>
<td>Number of buffer allocation failures</td>
</tr>
<tr>
<td>protocolError</td>
<td>Number of protocol errors</td>
</tr>
</tbody>
</table>

**LLC2_GET_SAP_STATS** The LLC2_GET_SAP_STATS command retrieves statistics related to a particular SAP. When sending the LLC2_GET_SAP_STATS command, the llc2GetSapStats structure is used:

```c
typedef struct llc2GetSapStats {
    uint_t ppa;
    uint_t cmd;
    uchar_t sap;
    uchar_t clearFlag;
    uchar_t state;
    uint_t numCons;
    ushort_t cons[LLC2_MAX_CONS];
    uint_t xidCmdSent;
    uint_t xidXidCmdRcvd;
    uint_t xidRspSent;
    uint_t xidRspRcvd;
} llc2GetSapStats_t;
```
```c
uint_t testCmdSent;
uint_t testCmdRcvd;
uint_t testRspSent;
uint_t testRspRcvd;
uint_t uiSent;
uint_t uiRcvd;
uint_t outOfState;
uint_t allocFail;
uint_t protocolError;
} llc2GetSapStats_t;
```

The members are:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppa</td>
<td>Physical Point of Attachment number</td>
</tr>
<tr>
<td>cmd</td>
<td>LLC2_GET_SAP_STATS</td>
</tr>
<tr>
<td>sap</td>
<td>SAP value</td>
</tr>
<tr>
<td>clearFlag</td>
<td>Clear counters flag. Set this to 0 to retrieve statistics and to 1 to reset all counters to 0.</td>
</tr>
<tr>
<td>state</td>
<td>SAP component state</td>
</tr>
<tr>
<td>numCons</td>
<td>Number of active connections in the cons array</td>
</tr>
<tr>
<td>cons</td>
<td>Array of active connection indexes</td>
</tr>
<tr>
<td>xidCmdSent</td>
<td>Number of XID commands sent</td>
</tr>
<tr>
<td>xidCmdRcvd</td>
<td>Number of XID responses received</td>
</tr>
<tr>
<td>xidRspSent</td>
<td>Number of XID responses sent</td>
</tr>
<tr>
<td>xidRspRcvd</td>
<td>Number of XID responses received</td>
</tr>
<tr>
<td>testCmdSent</td>
<td>Number of TEST commands sent</td>
</tr>
<tr>
<td>testCmdRcvd</td>
<td>Number of TEST commands received</td>
</tr>
<tr>
<td>testRspSent</td>
<td>Number of TEST responses sent</td>
</tr>
<tr>
<td>testRspRcvd</td>
<td>Number of TEST responses received</td>
</tr>
<tr>
<td>uiSent</td>
<td>Number of UI frames sent</td>
</tr>
<tr>
<td>uiRcvd</td>
<td>Number of UI frames received</td>
</tr>
<tr>
<td>outOfState</td>
<td>Number of invalid events received</td>
</tr>
<tr>
<td>allocFail</td>
<td>Number of buffer allocation failures</td>
</tr>
<tr>
<td>protocolError</td>
<td>Number of protocol errors</td>
</tr>
</tbody>
</table>
The `LLC2_GET_CON_STATS` command retrieves statistics related to a particular connection component. When sending the `LLC2_GET_CON_STATS` command, the `llc2GetConStats` structure is used:

```c
typedef struct llc2GetConStats {
    uint_t ppa;
    uint_t cmd;
    uchar_t sap;
    ushort_t con;
    uchar_t clearFlag;
    uchar_t stateOldest;
    uchar_t stateOlder;
    uchar_t stateOld;
    uchar_t state;
    ushort_t sid;
    disap_t rem;
    ushort_t flag;
    uchar_t dataFlag;
    uchar_t k;
    uchar_t vs;
    uchar_t vr;
    uchar_t nrRcvd;
    ushort_t retryCount;
    uint_t numToBeAcked;
    uint_t numToResend;
    uint_t macOutSave;
    uint_t macOutDump;
    uchar_t timerOn;
    uint_t iSent;
    uint_t iRcvd;
    uint_t frmrSent;
    uint_t frmrRcvd;
    uint_t rrSent;
    uint_t rrRcvd;
    uint_t rnrSent;
    uint_t rnrRcvd;
    uint_t rejSent;
    uint_t rejRcvd;
    uint_t sabmeSent;
    uint_t sabmeRcvd;
    uint_t uaSent;
    uint_t uaRcvd;
    uint_t discSent;
    uint_t outOfState;
    uint_t allocFail;
    uint_t protocolError;
    uint_t localBusy;
    uint_t remoteBusy;
    uint_t maxRetryFail;
    uint_t ackTimerExp;
    uint_t pollTimerExp;
    uint_t rejTimerExp;
    uint_t remBusyTimerExp;
    uint_t inactTimerExp;
    uint_t sendAckTimerExp;
} llc2GetConStats_t;
```
The members of the structure are:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppa</td>
<td>Physical Point of Attachment number</td>
</tr>
<tr>
<td>cmd</td>
<td>LLC2_GET_CON_STATS</td>
</tr>
<tr>
<td>sap</td>
<td>SAP value</td>
</tr>
<tr>
<td>con</td>
<td>Connection index</td>
</tr>
<tr>
<td>clearFlag</td>
<td>Clear counters flag. Set this to 0 to retrieve statistics and to 1 to reset all counters to 0.</td>
</tr>
<tr>
<td>stateOldest, stateOlder, stateOld, state</td>
<td>The four previous dlpi states of the connection</td>
</tr>
<tr>
<td>sid</td>
<td>SAP value and connection index</td>
</tr>
<tr>
<td>dl sap_t rem</td>
<td>Structure containing the remote MAC address and SAP</td>
</tr>
<tr>
<td>flag</td>
<td>Connection component processing flag</td>
</tr>
<tr>
<td>dataFlag</td>
<td>DATA_FLAG</td>
</tr>
<tr>
<td>k</td>
<td>transmit window size</td>
</tr>
<tr>
<td>vs</td>
<td>Sequence number of the next I-frame to send</td>
</tr>
<tr>
<td>vr</td>
<td>Sequence number of the next I-frame expected</td>
</tr>
<tr>
<td>nrRcvd</td>
<td>Sequence number of the last I-frame acknowledged by the remote node</td>
</tr>
<tr>
<td>retryCount</td>
<td>Number of timer expirations</td>
</tr>
<tr>
<td>numToBeAcked</td>
<td>Number of outbound I-frames to be acknowledged</td>
</tr>
<tr>
<td>numToResend</td>
<td>Number of outbound I-frames to be re-sent</td>
</tr>
<tr>
<td>macOutSave</td>
<td>Number of outbound I-frames held by the MAC driver to be saved on return to LLC2</td>
</tr>
<tr>
<td>macOutDump</td>
<td>Number of outbound I-frames held by the MAC driver to be dumped on return to LLC2</td>
</tr>
<tr>
<td>timerOn</td>
<td>Timer activity flag</td>
</tr>
<tr>
<td>iSent</td>
<td>Number of I-frames sent</td>
</tr>
<tr>
<td>iRcvd</td>
<td>Number of I-frames received</td>
</tr>
<tr>
<td>frm rSent</td>
<td>Number of frame rejects sent</td>
</tr>
<tr>
<td>frm rRcvd</td>
<td>Number of frame rejects received</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>rrSent</td>
<td>Number of RRs sent</td>
</tr>
<tr>
<td>rrRcvd</td>
<td>Number of RRs received</td>
</tr>
<tr>
<td>rrnrRcvd</td>
<td>Number of RNRs received</td>
</tr>
<tr>
<td>rejSent</td>
<td>Number of rejects sent</td>
</tr>
<tr>
<td>rejRcvd</td>
<td>Number of rejects received</td>
</tr>
<tr>
<td>sabmeSent</td>
<td>Number of SABMEs sent</td>
</tr>
<tr>
<td>sabmeRcvd</td>
<td>Number of SABMEs received</td>
</tr>
<tr>
<td>uaSent</td>
<td>Number of UAs sent</td>
</tr>
<tr>
<td>uaRcvd</td>
<td>Number of UAs received</td>
</tr>
<tr>
<td>discSent</td>
<td>Number of DISCs sent</td>
</tr>
<tr>
<td>outOfState</td>
<td>Number of invalid events received</td>
</tr>
<tr>
<td>allocFail</td>
<td>Number of buffer allocation failures</td>
</tr>
<tr>
<td>protocolError</td>
<td>Number of protocol errors</td>
</tr>
<tr>
<td>localBusy</td>
<td>Number of times in a local busy state</td>
</tr>
<tr>
<td>remoteBusy</td>
<td>Number of times in a remote busy state</td>
</tr>
<tr>
<td>maxRetryFail</td>
<td>Number of failures due to reaching maxRetry</td>
</tr>
<tr>
<td>ackTimerExp</td>
<td>Number of ack timer expirations</td>
</tr>
<tr>
<td>pollTimerExp</td>
<td>Number of P-timer expirations</td>
</tr>
<tr>
<td>rejTimerExp</td>
<td>Number of reject timer expirations</td>
</tr>
<tr>
<td>remBusyTimerExp</td>
<td>Number of remote busy timer expirations</td>
</tr>
<tr>
<td>inactTimerExp</td>
<td>Number of inactivity timer expirations</td>
</tr>
<tr>
<td>sendAckTimerExp</td>
<td>Number of send ack timer expirations</td>
</tr>
</tbody>
</table>

### FILES
/dev/llc2
/etc/llc2/default/llc2.? configuration files
(One file per ppa interface.)

### ATTRIBUTES
See attributes(5) for a description of the following attribute:
## llc2(7D)

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

SEE ALSO  
llc2_autoconfig(1), llc2_config(1), llc2(4)
lockstat(7D)

<table>
<thead>
<tr>
<th>NAME</th>
<th>lockstat – kernel lock statistics driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>The <code>lockstat</code> driver is the mechanism used by the <code>lockstat(1M)</code> command to extract kernel lock statistics. This is not a public interface.</td>
</tr>
<tr>
<td>FILES</td>
<td><code>/dev/lockstat</code> kernel lock statistics driver</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>lockstat(1M)</code></td>
</tr>
</tbody>
</table>
The lofi file driver exports a file as a block device. Reads and writes to the block device are translated to reads and writes on the underlying file. This is useful when the file contains a file system image. Exporting it as a block device through the lofi file driver allows normal system utilities to operate on the image through the block device (like fsyp(1M) fsck(1M), and mount(1M). This is useful for accessing CD-ROM and FAT floppy images. See lofiadm(1M) for examples.

File block device entries are contained in /dev/lofi, while /dev/rlofi contains the character (or raw) device entries. Entries are in the form of decimal numbers which are assigned through lofiadm(1M). When created, these device entries are owned by root, in group sys, and have permissions 0600. While ownership, group, and permission settings can be altered, there are possible ramifications. See lofiadm(1M) for more information.

FILES
/dev/lofi
   Master control device
/dev/lofi/n
   Block device for file n
/dev/rlofi/n
   Character device for file n
/kernel/drv/lofi
   32-bit driver
/kernel/drv/lofi.conf
   Driver configuration file. (Should not be altered.)
/kernel/drv/sparcv9/lofi
   64-bit 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>Availability</td>
<td>SUNWcsr, SUNWcarx.u</td>
</tr>
</tbody>
</table>

SEE ALSO
lofiadm(1M), fsck(1M), fsyp(1M), mount(1M), newfs(1M), attributes(5)

NOTES
Just as you would not directly access a disk device that has mounted file systems, you should not access a file associated with a block device except through the lofi file driver.

For compatibility purposes, a raw device is also exported along with the block device. For example, newfs(1M) requires one.
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
Files can be modified on a read-only loopback mounted file system, and 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 arg1 ...)
```

Required definitions are contained in `<sys/strlog.h>`, `<sys/log.h>`, and `<sys/syslog.h>`. `mid` is the STREAMS module id number for the module or driver submitting the log message. `sid` is an internal sub-id number usually used to identify a particular minor device of a driver. `level` is a tracing level that allows for selective screening out of low priority messages from the tracer. `flags` are any combination of `SL_ERROR` (the message is for the error logger), `SL_TRACE` (the message is for the tracer), `SL_CONSOLE` (the message is for the console logger), `SL_FATAL` (advisory notification of a fatal error), and `SL_NOTIFY` (request that a copy of the message be mailed to the system administrator). `fmt` is a `printf(3C)` style format string, except that `%s`, `%e`, `%E`, `%g`, and `%G` conversion specifications are not handled. Up to `NLOGARGS` (in this release, three) numeric or character arguments can be provided.

#### User Interface
log is implemented as a cloneable device, it clones itself without intervention from the system clone device. Each open of `/dev/log` obtains a separate stream to log. In order to receive log messages, a process must first notify log whether it is an error logger, trace logger, or console logger using a STREAMS I_STR ioctl call (see below). For the console logger, the I_STR ioctl has an `ic_cmd` field of I_CONSLOG, with no accompanying data. For the error logger, the I_STR ioctl has an `ic_cmd` field of I_ERRLOG, with no accompanying data. For the trace logger, the ioctl has an `ic_cmd` field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct `trace_ids` elements.
```
struct trace_ids {
   short ti_mid;
   short ti_mid;
   char ti_level;
};
```
Each trace_ids structure specifies a mid, sid, and level from which messages will be accepted. strlog(9F) will accept messages whose mid and sid exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of −1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

Once the logger process has identified itself using the ioctl call, log will begin sending up messages subject to the restrictions noted above. These messages are obtained using the getmsg(2) function. The control part of this message contains a log_ctl structure, which specifies the mid, sid, level, flags, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, a sequence number, and a priority. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of log messages can be determined.

```
struct log_ctl {
    short mid;
    short sid;
    char level; /* level of message for tracing */
    short flags; /* message disposition */
#if defined(_LP64) || defined(_I32LPx)
    clock32_t ltime; /* time in machine ticks since boot */
    time32_t ttime; /* time in seconds since 1970 */
#else
    clock_t ltime;
    time_t ttime;
#endif
    int seq_no; /* sequence number */
    int pri; /* priority = (facility|level) */
};
```

The priority consists of a priority code and a facility code, found in <sys/syslog.h>. If SL_CONSOLE is set in flags, the priority code is set as follows: If SL_WARN is set, the priority code is set to LOG.WARNING; If SL_FATAL is set, the priority code is set to LOG.CRIT; If SL_ERROR is set, the priority code is set to LOG.ERR; If SL_NOTE is set, the priority code is set to LOG.NOTICE; If SL_TRACE is set, the priority code is set to LOG.DEBUG; If only SL_CONSOLE is set, the priority code is set to LOG.INFO. Messages originating from the kernel have the facility code set to LOG_KERN. Most messages originating from user processes will have the facility code set to LOG_USER.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.
A process may also send a message of the same structure to `log`, even if it is not an error or trace logger. The only fields of the `log_ctl` structure in the control part of the message that are accepted are the `level`, `flags`, and `pri` fields; all other fields are filled in by `log` before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to `NLOGARGS`) must be packed, 32-bits each, on the next 32-bit boundary following the end of the format string.

`ENXIO` is returned for `I_TRCLOG` `ioctl`s without any `trace_ids` structures, or for any unrecognized `ioctl` calls. The driver silently ignores incorrectly formatted `log` messages sent to the driver by a user process (no error results).

Processes that wish to write a message to the console logger may direct their output to `/dev/conslog`, using either `write(2)` or `putmsg(2)`.

The following driver configuration properties may be defined in the `log.conf` file.

msgid=1 If `msgid=1`, each message will be preceded by a message ID as described in `syslogd(1M)`.

msgid=0 If `msgid=0`, message IDs will not be generated. This property is unstable and may be removed in a future release.

**EXAMPLE 1** `I_ERRLOG` registration.

```c
struct strioctl ioc;
ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0; /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;
ioctl(log, I_STR, &ioc);
```

**EXAMPLE 2** `I_TRCLOG` registration.

```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 */
ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;
ioctl(log, I_STR, &ioc);
```

Example of submitting a `log` message (no arguments):

```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);
```
EXAMPLE 2 I TRCLOG registration. (Continued)

ctl.buf = (char *)&lc;
dat.len = dat maxlen = strlen(message);
dat.buf = message;
lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;
putmsg(log, &ctl, &dat, 0);

FILES /dev/log Log driver.
/dev/conslog Write only instance of the log driver, for console logging.
/kernel/drv/log.conf Log configuration file.

SEE ALSO strace(1M), strerror(1M), intro(3), getmsg(2), ioctl(2), putmsg(2), write(2), printf(3C), 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>IA</td>
</tr>
</tbody>
</table>

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

SYNOPSIS

```
#include <sys/bpp_io.h>
fd = open("/dev/lp", flags);
```

DESCRIPTION

The `lp` driver provides the interface to the parallel ports used by printers for IA based systems. The `lp` driver is implemented as a STREAMS device.

IOCTLS

- **BPPIOC_TESTIO**: Test transfer readiness. This command checks to see if a read or write transfer would succeed based on pin status. If a transfer would succeed, 0 is returned. If a transfer would fail, -1 is returned, and `errno` is set to `EIO`. The error status can be retrieved using the `BPPIOC_GETERR` ioctl() call.

- **BPPIOC_GETERR**: Get last error status. The argument is a pointer to a `struct bpp_error_status`. See below for a description of the elements of this structure. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a `read(2)` or `write(2)` call, or the status of the bits at the most recent `BPPIOC_TESTIO ioctl(2)` call. The application can check transfer readiness without attempting another transfer using the `BPPIOC_TESTIO ioctl()`.

Error Pins Structure

This structure and symbols are defined in the include file `<sys/bpp_io.h>`:

```
struct bpp_error_status {
    char timeout_occurred; /* Not use */
    char bus_error; /* Not use */
    uchar_t pin_status; /* Status of pins which could cause an error */
    */
    #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 IA `lp` driver.

ERRORS

- **EIO**: A `BPPIOC_TESTIO ioctl()` call is attempted while a condition exists that would prevent a transfer (such as a peripheral error).

- **EINVAL**: An `ioctl()` is attempted with an invalid value in the command argument.

FILES

```
/platform/i86pc/kernel/drv/lp.conf
```

configuration file for `lp` driver

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:
### ATTRIBUTE TYPE | ATTRIBUTE VALUE
---|---
Architecture | IA

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

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

```c
#include <sys/types.h>
#include <fcntl.h>
#include <visual.h>
#include <sys/litem.h>
```

DESCRIPTION | The litem driver provides a general-purpose ANSI interface to the system console device. litem is a layered device driver which on one side provides the kernel with a consistent interface to the system console device (and therefore 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 litem that the resolution of the underlying framebuffer has been changed. litem 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/litem/*

SEE ALSO | ioctl(2), visual_io(7I)
m64(7D)

NAME
m64 – PGX, PGX24, and PGX64 frame buffers device driver

DESCRIPTION
The m64 driver is the Sun PGX graphics accelerator device driver.

FILES
/dev/fbs/m64\fIn      Device special file

SEE ALSO
m64config(1M)
NAME
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.

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.
mhd(7i)

<table>
<thead>
<tr>
<th>NAME</th>
<th>mhd – multihost disk control operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>#include &lt;sys/mhd.h&gt;</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The mhd ioctl(2) control access rights of a multihost disk, using disk reservations on the disk device. The stability level of this interface (see attributes(5)) is Evolving, thus, use of this interface should be limited and users of the interface will find that it is subject to change. The mhd ioctls fall into two major categories: • ioctls for non-shared multihost disks, and • ioctls for shared multihost disks. One ioctl, MHIOCENFAILFAST, is applicable to both non-shared and shared multihost disks. It is described after the first two categories. All the ioctls require root privilege. For all of the ioctls, the caller should obtain the file descriptor for the device by calling open(2) with the O_NDELAY flag; without the O_NDELAY flag, the open may fail due to another host already having a conflicting reservation on the device. Some of the ioctls below permit the caller to forcibly clear a conflicting reservation held by another host, however, in order to call the ioctl, the caller must first obtain the open file descriptor. Non-shared multihost disks Non-shared multihost disk ioctls consist of MHIOCTKOWN, MHIOCRELEASE, HIOCSTATUS, and MHIOCQRESERVE. These ioctl requests control the access rights of non-shared multihost disks. A non-shared multihost disk is one that supports serialized, mutually exclusive I/O mastery by the connected hosts. This is in contrast to the shared-disk model, in which concurrent access is allowed from more than one host (see below). A non-shared multihost disk can be in one of two states: • exclusive access state, where only one connected host has I/O access, or • 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.</td>
</tr>
</tbody>
</table>
Each multihost disk driver views the machine on which it’s running as the "local host"; each views all other machines as "remote hosts". For each I/O or ioctl request, the requesting host is the local host.

Note that the non-shared ioctls are designed to work with SCSI-2 disks. The SCSI-2 RESERVE/RELEASE command set is the underlying hardware facility in the device that supports the non-shared ioctls.

The function prototypes for the non-shared ioctls are:

```c
ioctl(fd, MHIOCTKOWN, (struct mhioctkown *)tkown);
ioctl(fd, MHIOCRELEASE);
ioctl(fd, MHIOCSTATUS);
ioctl(fd, MHIOCQRESERVE);
```

- **MHIOCTKOWN**: Forcefully acquires exclusive access rights to the multihost disk for the local host. Revokes all access rights to the multihost disk from remote hosts. Causes the disk to enter the exclusive access state.

  Implementation Note: Reservations (exclusive access rights) broken via random resets should be reinstated by the driver upon their detection, for example, in the automatic probe function described below.

- **MHIOCRELEASE**: Relinquishes exclusive access rights to the multihost disk for the local host. On success, causes the disk to enter the non-exclusive access state.

- **MHIOCSTATUS**: Probes a multihost disk to determine whether the local host has access rights to the disk. Returns 0 if the local host has access to the disk, 1 if it doesn’t, and -1 with errno set to EIO if the probe failed for some other reason.

- **MHIOCQRESERVE**: Issues, simply and only, a SCSI-2 Reserve command. If the attempt to reserve fails due to the SCSI error Reservation Conflict (which implies that some other host has the device reserved), then the ioctl will return -1 with errno set to EACCES. The **MHIOCQRESERVE** ioctl does NOT issue a bus device reset or bus reset prior to attempting the SCSI-2 reserve command. It also does not take care of re-instating reservations that disappear due to bus resets or bus device resets; if that behavior is desired, then the caller can call **MHIOCTKOWN** after the **MHIOCQRESERVE** has returned success. If the device does not support the SCSI-2 Reserve command, then the ioctl returns -1 with errno set to ENOTSUP. The **MHIOCQRESERVE** ioctl is intended to be used by high-availability or clustering software for a "quorum" disk, hence, the "Q" in the name of the ioctl.

**Shared Multihost Disks**

Shared multihost disks ioctls control access to shared multihost disks. The ioctls are merely a veneer on the SCSI-3 Persistent Reservation facility. Therefore, the underlying
The function prototypes and descriptions for the shared multihost ioctls are as follows:

```c
ioctl(fd, MHIOCGRP_INKEYS, (mhioc_inkeys_t*)k);
   Issues the SCSI-3 command Persistent Reserve In Read Keys to the device. On input, the field k->li should be initialized by the caller with k->li.listsize reflecting how big of an array the caller has allocated for the k->li.list field and with k->li.listlen == 0. On return, the field k->li.listlen is updated to indicate the number of reservation keys the device currently has: if this value is larger than k->li.listsize then that indicates that the caller should have passed a bigger k->li.list array with a bigger k->li.listsize. The number of array elements actually written by the callee into k->li.list is the minimum of k->li.listlen and k->li.listsize. The field k->generation is updated with the generation information returned by the SCSI-3 Read Keys query. If the device does not support SCSI-3 Persistent Reservations, then this ioctl returns -1 with errno set to ENOTSUP.

ioctl(fd, MHIOCGRP_INRESVS,(mhioc_inresvs_t*)r);
   Issues the SCSI-3 command Persistent Reserve In Read Reservations to the device. Remarks similar to MHIOCGRP_INKEYS apply to the array manipulation. If the device does not support SCSI-3 Persistent Reservations, then this ioctl returns -1 with errno set to ENOTSUP.

ioctl(fd, MHIOCGRP_REGISTER,(mhioc_register_t*)r);
   Issues the SCSI-3 command Persistent Reserve Out Register. The fields of structure r are all inputs; none of the fields are modified by the ioctl. The field r->aptpl should be set to true to specify that registrations and reservations should persist across device power failures, or to false to specify that registrations and reservations should be cleared upon device power failure; true is the recommended setting. The field r->oldkey is the key that the caller believes the device may already have for this host initiator; if the caller believes that that this host initiator is not already registered with this device, it should pass the special key of all zeros. To achieve the effect of unregistering with the device, the caller should pass its current key for the r->oldkey field and an r->newkey field containing the special key of all zeros. If the device returns the SCSI error code Reservation Conflict, this ioctl returns -1 with errno set to EACCES.

ioctl(fd, MHIOCGRP_RESERVE,(mhioc_resv_desc_t*)r);
   Issues the SCSI-3 command Persistent Reserve Out Reserve. The fields of structure r are all inputs; none of the fields are modified by the ioctl. If the device returns the SCSI error code Reservation Conflict, this ioctl returns -1 with errno set to EACCES.

ioctl(fd, MHIOCGRP_PREEMPTANDABORT,(mhioc_preemptandabort_t*)r);
   Issues the SCSI-3 command Persistent Reserve Out Preempt-And-Abort. The fields of structure r are all inputs; inputs; none of the fields are modified by the ioctl. The key of the victim host is specified by the field r->victim_key. The field
```
r->resvdesc supplies the preempter's key and the reservation that it is
requesting as part of the SCSI-3 Preempt-And-Abort command. If the device
returns the SCSI error code Reservation Conflict, this ioctl returns -1 with errno
set to EACCES.

```c
ioctl(fd, MHIOCGRP_PREEMPT, (mhioc_preemptandabort_t *)&r);
```
Similar to MHIOCGRP_PREEMPTANDABORT, but instead issues the SCSI-3 command
Persistent Reserve Out Preempt.

```c
ioctl(fd, MHIOCGRP_CLEAR, (mhioc_resv_key_t *)&r);
```
Issues the SCSI-3 command Persistent Reserve Out Clear. The input parameter r is
the reservation key of the caller, which should have been already registered with
the device, by an earlier call to MHIOCGRP_REGISTER.

For each device, the non-shared ioctls should not be mixed with the Persistent Reserve
Out shared ioctls, and vice-versa, otherwise, the underlying device is likely to return
errors, because SCSI does not permit SCSI-2 reservations to be mixed with SCSI-3
reservations on a single device. It is, however, legitimate to call the Persistent Reserve
In ioctls, because these are query only. Issuing the MHIOCGRP_INKEYS ioctl is the
recommended way for a caller to determine if the device supports SCSI-3 Persistent
Reservations (the ioctl will return -1 with errno set to ENOTSUP if the device does
not).

**MHIOCENFAILFAST ioctl**
The MHIOCENFAILFAST ioctl is applicable for both non-shared and shared disks, and
may be used with either the non-shared or shared ioctls.

```c
ioctl(fd, MHIOCENFAILFAST, (unsigned int *)millisecs);
```
Enables or disables the failfast option in the multihost disk driver and enables or
disables automatic probing of a multihost disk, described below. The argument is
an unsigned integer specifying the number of milliseconds to wait between
executions of the automatic probe function. An argument of zero disables the
failfast option and disables automatic probing. If the MHIOCENFAILFAST ioctl is
never called, the effect is defined to be that both the failfast option and automatic
probing are disabled.

**Automatic Probing**
The MHIOCENFAILFAST ioctl sets up a timeout in the driver to periodically schedule
automatic probes of the disk. The automatic probe function works in this manner: The
driver is scheduled to probe the multihost disk every n milliseconds, rounded up to
the next integral multiple of the system clock's resolution. If

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

If the driver makes this discovery outside the timeout function, especially during a
read or write operation, it is imperative that it panic the system then as well.

**RETURN VALUES**
Each request returns -1 on failure and sets errno to indicate the error.
mhd(7i)

**EPERM**
Caller is not root.

**EACCES**
Access rights were denied.

**EIO**
The multihost disk or controller was unable to successfully complete the requested operation.

**EOPNOTSUP**
The multihost disk does not support the operation. For example, it does not support the SCSI-2 Reserve/Release command set, or the SCSI-3 Persistent Reservation command set.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWheaw</td>
</tr>
<tr>
<td>Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`ioctl(2), open(2), attributes(5)open(2)`

**NOTES**
The `ioctl`s for shared multihost disks and the `MHIOCQRESERVE` `ioctl` are currently implemented only for SPARC and only for the following disk device drivers: `sd(7D), SSD(7D)`. 
mixer(7I)

NAME mixer – audio mixer audio personality module interface

SYNOPSIS #include <sys/mixer.h>

OVERVIEW The audio mixer extends the audio(7I) interface, allowing more than one process to play or record audio at the same time. Understanding the audio(7I) interface thoroughly is a prerequisite to understanding the mixer(7I) interface.

Backward Compatibility It is possible to disable the mixing function and return to 100% backward compatibility with the audio(7I) interface. These two modes of operation are referred to as the mixer mode and the compatible mode. This is done by using either the mixerctl(1) or sdtaudioccontrol(1) applications, or by editing the audio driver.conf file and then unloading and reloading the driver.

Multi-Stream Codecs The audio mixer supports multi-stream Codecs. Examples of these Codecs are the Crystal Semiconductor 4410/4422 and the Aureal 8820/8830. These devices have DSP engines that provide sample rate conversion and other features. Each play/record channel is mapped to an individual channel straight into the Codec. The audio mixer does not perform sample rate or encoding conversion. (See below). However, the programming interfaces remain the same and applications cannot distinguish between multi-stream Codec and traditional Codec.

Buffer Size An application can use the audio_info_t structure to set the size of the play/record buffers. As with the audio(7I) interface, the audio mixer does not support changing of the play buffer. (The audio driver takes sound samples as they are needed, regardless of how many are delivered with each write.)

The audio mixer does support changing of the record buffer. When captured by the audio driver, buffer size bytes are sent to the application to read.

AUDIO FORMATS See the audio(7I) manual page for a brief discussion of audio formats. To mix the various audio streams, the audio mixer must convert all audio formats to a common format. The following describes how the audio mixer deals with these different components.

Sample Rate When /dev/audio is opened, the initial sample rate is 8KHz, as defined in audio(7I).

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

Sample rate conversion can be a compute-intensive operation, depending on the number of channels and a device’s sample rate. For example, an 8KHz signal can be easily converted to 48KHz, requiring a low cost up sampling by 6. However, converting from 44.1KHz to 48KHz is compute intensive because it must be up sampled by 160 and then down sampled by 147. (This is only done using integer multipliers.)
Applications can greatly reduce the impact of sample rate conversion by carefully picking the sample rate. Applications should always use the highest sample rate the device supports. An application can also do its own sample rate conversion (to take advantage of floating point and accelerated instruction) or use small integers for up and down sampling.

In compatible mode, the audio mixer programs the Codec to the sample rate set by the application to avoid incurring any sample rate conversion overhead. If the Codec cannot support different play and record sample rates, the `AUDIO_SETINFO ioctl(2)` will fail.

When `/dev/audio` is opened, initial encoding and precision is 8-bit μ-Law (as in the Greek letter mu). (As defined in `audio(7I)`)

In mixer mode, the audio mixer supports formats in the following precisions:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed Linear PCM</td>
<td>16-bit</td>
<td>Mono or Stereo</td>
</tr>
<tr>
<td>Signed Linear PCM</td>
<td>8-bit</td>
<td>Mono or Stereo</td>
</tr>
<tr>
<td>μ-Law</td>
<td>8-bit</td>
<td>Mono or Stereo</td>
</tr>
<tr>
<td>A-Law</td>
<td>8-bit</td>
<td>Mono or Stereo</td>
</tr>
</tbody>
</table>

The audio mixer converts all audio streams to 16-bit Linear PCM before mixing. After mixing, conversion is made to the best possible Codec format. The conversion process is not compute intensive and audio applications can choose the encoding format the best meets its needs.

In compatibility mode, the audio mixer sets the Codec to the encoding and precision set by the application. If the Codec cannot support different play and record encodings or precisions, the `AUDIO_SETINFO ioctl(2)` will fail.

When `/dev/audio` is opened, the number of initial channels is 1, left channel mono. (As defined in `audio(7I)`). Most Codecs play or record mono audio on the left channel.

In mixer mode, the audio mixer sets the Codec to the maximum number of channels supported. If a mono signal is played or recorded, it is mixed on the first (usually the left) channel only. Silence is mixed on the remaining channels.

In compatible mode, the audio mixer sets the Codec to the number of channels set by the application. If the Codec cannot support a different number of play and record channels, the `AUDIO_SETINFO ioctl(2)` will fail.

The device `/dev/audio` is a device driver that dispatches audio requests to the appropriate underlying audio personality module. The audio driver is implemented as a STREAMS driver. To record audio input, applications open(2) the `/dev/audio`
Opening the Audio Device

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.

In mixer mode, the audio device is no longer treated as an exclusive resource. Each process may open the audio device once unless the process has made an `AUDIO_MIXER_MULTIPLE_OPEN ioctl(2)`. See below for details.

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

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

Upon the initial `open()` of the audio channel, the audio mixer sets the data format of the audio channel to the default state of 8-bit, 8KHz, mono μ-Law data. If the audio device does not support this configuration, it informs the audio mixer of the initial configuration. Audio applications should explicitly set the encoding characteristics to match the audio data requirements, and not depend on the default configuration. See the `audio(7)` manual page for details on how the audio mixer behaves when in compatible mode.

Recording Audio Data

The `read(2)` 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 also be set to a non-blocking mode, where `read()` completes immediately but may return fewer bytes than requested. See 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. Because 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_info_t` information structure. (See `audio(7)`). All unread input data in the STREAMS queue may be discarded by using the `I_FLUSH STREAMS ioctl(2)` (see `streamio(7I)`). When changing record parameters, the input stream should first be paused and flushed before the change. Otherwise, subsequent reads may return samples in the old format, followed by samples in the new format.

Input data accumulates in STREAMS buffers rapidly. For 8-bit, 8 KHz, mono μ-Law data, it accumulates at 8000 bytes per second. If a device is configured for 16-bit linear or higher sample rates, it accumulates even faster. If the application that consumes the data is unable to meet the input data rate, the STREAMS queue may become full. When this happens, the `record.error` flag is set in the `audio_info_t` information...
structure and input sampling ceases until there is room for additional data, resulting in a data stream discontinuity. To prevent this, audio recording applications should open the audio device when they are ready to begin reading data and not at the start of extensive initialization.

The write(2) system call copies data from an application’s buffer to the STREAMS output queue. Ordinarily, write() blocks until the entire user buffer is transferred. The device may alternatively be set to a non-blocking mode, in which case write() completes immediately, but may transfer fewer bytes than requested. (See the write(2) manual page for a complete description of this behavior).

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_info_t information structure (see below) is incremented.

The final close(2) of the audio device file descriptor hangs until all of the process’ remaining audio output has drained. If a signal interrupts the close(), or if the process exits without closing the audio device, any remaining data queued for audio output is flushed and the audio 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_info_t structure. Queued output data may be discarded by using the I_FLUSH STREAMS ioctl. (See streamio(7I).)

Output data will be played from the STREAMS buffers at a default rate of 8000 bytes per second for µ-Law, A-Law, or 8-bit PCM dat. (And at a faster rate for 16-bit linear data or higher sampling rates). If the output queue becomes empty, the play.error flag is set in the audio_info_t structure and output is stopped until additional data is queued. 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 bad data will be thrown away. Additional writes are allowed.

The I_SETSIG STREAMS ioctl (see streamio(7I)) 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 a background audio stream.

The /dev/audioctl pseudo-device enables an application to modify characteristics of the audio device while it is being used by an unrelated process. Any number of processes may open /dev/audioctl pseudo device simultaneously. However, read() and write() system calls are ignored.
Note: 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. (See streamio(7I)). Such processes receive a SIGPOLL signal when any of the following events occur:

- AUDIO_SETUPINFO, AUDIO_MIXERCTL_SETINFO, AUDIO_MIXERCTL_SET_CHINFO, or AUDIO_MIXERCTL_SET_MODE ioctl() has altered the device state.
- Input overflow or output underflow has occurred.
- End-of-file record (zero-length buffer) has been processed on output.
- open() or close() of /dev/audio has altered the device state.
- External event (such as speakerbox volume control) has altered the device state.

The audio mixer implements all the ioctl()s defined in audio(7I) and uses the audio_prinfo_t, audio_info_t, and audio_device_t structures. See the audio(7I) manual page for details on these ioctl()s and structures. The audio mixer also uses the data structures described below.

The state of the audio device may be polled or modified using the AUDIO_MIXERCTL_GETINFO and AUDIO_MIXERCTL_SETINFO ioctl commands.

```c
typedef struct am_control {
    audio_info_t dev_info; /* the audio device's state */
    int8_t ch_open[1]; /* variable sized array of open chs */
} am_control_t;
```

See CODE EXAMPLES for sample code on how to use this structure and the related macro AUDIO_MIXER_CTL_STRUCT_SIZE(num_ch).

The following structure is used by the AUDIO_MIXER_GET_SAMPLE_RATES ioctl to get a list of all the supported sample rates.

```c
typedef struct am_sample_rates {
    uint_t type; /* play or capture */
    uint_t flags;
    uint_t num_samp_rates; /* number of elements in samp_rates[] */
    uint_t samp_rates[1]; /* variable sized array of sample rates */
} am_sample_rates_t;
```

```c
#define AUDIO_PLAY 0 /* type */
#define AUDIO_RECORD 1
```

```c
#define MIXER_SR_LIMITS 0x00000001 /* flags */
```

See CODE EXAMPLES for example code on how to use this structure and the related macro AUDIO_MIXER_SAMP_RATES_STRUCT_SIZE(num_srs).
When in mixer mode the `audio_info_t` structure's `sw_features_enabled` field will have `AM_MIXER` set. When in compatibility mode, the `AM_MIXER` bit will be cleared.

The defines for the `sw_features` and the `sw_features_enabled` fields are:
```c
#define AM_MIXER 0x00000001 /* mixer is present/enabled */
```

All of the `streamio(7I)` ioctl commands may be issued for the `/dev/audio` and `/dev/audiocntl` devices. `I_SETSIG ioctl` may be issued for `/dev/audiocntl` to enable the notification of audio status changes as described above.

Except for `AUDIO_MIXER_GET_SAMPLE_RATE`, `AUDIO_MIXERCTL_GET_MODE`, and `AUDIO_MIXERCTL_SET_MODE`, these `ioctl()`s are valid only in mixer mode. Using them in compatible mode will cause an `EINVAL` error to be returned.

`AUDIO_MIXER_MULTIPLE_OPEN`

The argument is ignored. This command allows an individual process to open `/dev/audio` more than once for play or record. This feature is useful for mixing panels that may be controlling multiple audio streams.

`AUDIO_MIXER_SINGLE_OPEN`

The argument is ignored. This command returns `/dev/audio` back to an exclusive access device on per process basis after an `AUDIO_MIXER_MULTIPLE_OPEN` `ioctl()` has been executed. This `ioctl()` will fail if more than one play or record stream is open.

`AUDIO_MIXER_GET_SAMPLE_RATES`

The argument is a pointer to an `am_sample_rates_t` structure. This command gets a list of supported sample rates for either play or record for the audio mixer's current mode. It is legal for the supported sample rates to be different for mixer mode vs compatible mode. The `type` field must be set to either `AUDIO_PLAY` or `AUDIO_RECORD` to get a list of either play or record sample rates, respectively. Setting both or neither is an error. The `num_samp_rates` field is set to the number of sample rates that the `samp_rates[]` array may hold. When the `ioctl` returns, `num_samp_rates` will be set either to the number of sample rates in the array `samp_rates[]`, or the total number of sample rates available if there are more then the array can hold. In the former case, there are `num_samp_rates` valid sample rates in the array. In the later case, all the elements of the array have valid sample rates, but there are more available. The size of the array should be increased to get all available sample rates. If the `flags` field has `MIXER_SR_LIMITS` flag set, the return sample rates are the lowest and the highest rates possible, with all sample rates in-between being legal. Some Codecs that have DSP engines on them have this capability.

`AUDIO_MIXERCTL_GETINFO`

The argument is a pointer to an `am_control_t` structure. This command gets device and channel state information. The `dev_info` field contains the state of the hardware device. It provides a convenient way to determine the hardware's state.
The `ch_open` array is used to specify which channels are open and which are closed. Open channels are non-zero, closed channels are set to zero, (where the channel number corresponds to the array index). The number of elements in the `ch_open` array may change over time and a macro is provided to allocate the correct amount of space. The MACROS section below provides more information.

**AUDIO_MIXERCTL_SETINFO**

The argument is a pointer to an `am_control_t` structure. This command sets the device state but cannot modify any channel’s state. The `dev_info` field is used to set the device state. However, there are several limitations. Only the `gain`, `balance`, `port` and `pause` for play and record and `monitor_gain` and `output_muted` fields may be modified. (Modifying other fields would interfere with how the audio mixer programs the audio device.) The `ch_open` array is not used when setting the audio device and may be set to a size of one.

**AUDIO_MIXERCTL_GET_CHINFO**

The argument is a pointer to an `audio_channel_t` structure. This command gets a channel’s state information. To enable the audio mixer to determine channel information, set the `ch_number` field before making the `ioctl()` call. The `info_size` field must be set to the size of the `audio_info_t` structure. The `*info` field must point to an `audio_info_t` structure. When the `ioctl()` returns, the `pid` field should be checked. If it is set to 0, the remaining data in the `audio_channel_t` structure is invalid because the channel has not been allocated. The `dev_type` field describes the type of channel; the `*info` pointer points to a buffer where the `audio_info_t` structure for the audio channel is populated.

**AUDIO_MIXERCTL_SET_CHINFO**

The argument is a pointer to an `audio_channel_t` structure. This command sets a channel’s state information. To enable the audio mixer to determine which channel to set, set the `ch_number` field before making the `ioctl()` call. The `info_size` field must be set to the size of the `audio_info_t` structure. The `*info` field must point to an `audio_info_t` structure. When the `ioctl()` returns, the `pid` will contain the process ID of the process that has the channel open and `dev_type` will contain the type of the device. If `pid` is 0 (zero), then the channel is not open. The `*info` pointer points to an `audio_info_t` structure which is used to program the state of the channel.

**AUDIO_MIXERCTL_GET_MODE**

The argument is a pointer to an integer that contains the audio mixer mode when it returns. It will be set to either `AM_MIXER_MODE` for mixer mode or `AM_COMPAT_MODE` for compatibility mode.

**AUDIO_MIXERCTL_SET_MODE**

The argument is a pointer to an integer that contains the audio mixer mode to be set. It must be set to either `AM_MIXER_MODE` or `AM_COMPAT_MODE`. The audio mixer may be set to mixer mode at any time, but can only be set to compatible mode when there is a single read/write open within one process, or a single read process and a single write process. Otherwise the `ioctl()` will fail. Because the Codec is being reprogrammed to a different data format, there may be brief pause
or burst of noise when the mode changes. This can be eliminated by pausing the
input and output or by closing all streams before changing modes. The
mixerctl(1) command may be used to change the audio mixer’s mode.

MACROS

The following macro is used to determine how large an am_control_t structure is
when it points to an audio_info_t structure.

\[
\text{AUDIO_MIXER_CTL_STRUCT_SIZE}(\text{num} \_\text{ch})
\]

Where \text{num} \_\text{ch} is the number of channels the device supports. The number of
channels can be determined using the \text{AUDIO_GET_NUM_CHS} ioctl().

This macro is used when allocating an am_sample_rates_t structure.

\[
\text{AUDIO_MIXER_SAMP_RATES_STRUCT_SIZE}(\text{num} \_\text{srs})
\]

Where \text{num} \_\text{srs} is the number of samples rates requested.

CODE EXAMPLES

The following examples illustrate how these new data structures and ioctls can be
used.

Example 1

The following code demonstrates how to use the audio support and the audio
mixer ioctl(1)s to get state information on /dev/audio.

```c
audio_channel_t ch;
audio_info_t info;
am_control_t *ctl;
int num;
err = ioctl(audio_fd, AUDIO_GET_NUM_CHS, &num);

c = (am_control_t *)malloc(AUDIO_MIXER_CTL_STRUCT_SIZE(num));
err = ioctl(audio_fd, AUDIO_MIXERCTL_GETINFO, c);

ch.info = &info;
ch.info_size = sizeof (audio_info_t);
for (i = 0; i < num; i++) {
    if (c->ch_open[i] != 0) {
        ch.ch_number = i;
        if (ioctl(audio_fd, AUDIO_MIXERCTL_GET_CHINFO, &ch) < 0) {
            printf("Channel #%d isn’t an audio/audioctl device", i);
        } else {
            printf("Ch# %d, PID = %d, Type = %d\n", i, ch.pid, ch.dev_type);
            
        
    }

Example 2

The following code demonstrates how to use the
\text{AUDIO_MIXER_GET_SAMPLE_RATES} ioctl to get the number of supported play
sample rates. It also shows how to deal with allocating a samp_rates[] array that is
too small.
```c
#define LARGE_NUMBER 10000;
int num;
for (num = 4; num < LARGE_NUMBER; num += 2) {
    sr = (am_sample_rates_t *)malloc(AUDIO_MIXER_SAMP_RATES_STRUCT_SIZE(num));
    sr->num_samp_rates = num;
    sr->type = AUDIO_PLAY;
    err = ioctl(audio_fd, AUDIO_MIXER_GET_SAMPLE_RATES, sr);
    if (sr->num_samp_rates <= num) {
        break;
    }
    free(sr);
}
(void) printf("Supported play sample rates:\n");
for (i = 0; i < sr->num_samp_rates; i++) {
    (void) printf(" %d\n", sr->samp_rates[i]);
}
```

**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.
- **ENOMEM** Memory was not available to be allocated for the channel.
- **EINTR**  The requested play or record access is busy and a signal interrupted the open() request.
- **EIO**    There has been an error opening the device. An error message is printed on the console explaining the failure.

An ioctl() will fail if:

- **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.
- **EINVAL** The parameter changes requested in the AUDIO_SETINFO ioctl are invalid or are not supported by the device.
- **EIO**    There has been an error with the ioctl(). An error message is printed on the console explaining the failure.
- **ENOMEM** The ioctl() failed because memory couldn't be allocated.
- **EPERM**  The audio mixer is in compatible mode and one of the new ioctl()s was used. They are supported only in mixer mode.
**mixer(7I)**

**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/audiocctl` symbolic link to the control device for `/dev/audio`
- `/dev/sound/0` first audio device in the system
- `/dev/sound/0cctl` audio control device for `/dev/sound/0`
- `/dev/sound/x` additional audio devices
- `/dev/sound/xct1` audio control device for `/dev/sound/x`

**ATTRIBUTES**
See attributes(5) for a description 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>
<tr>
<td>Availability</td>
<td>SUNWaudd, SUNWauddx, SUNWaudh</td>
</tr>
<tr>
<td>Stability Level</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**
mixerctl(1), close(2), fcntl(2), ioctl(2), open(2), poll(2), read(2), write(2), system(4), audiocs(7D), usb_ac(7D), audio_support(7I), streamio(7I)

**BUGS**
Due to a feature of the STREAMS implementation, programs that are terminated or exit without closing the audio device may hang for a short period while audio output drains. In general, programs that produce audio output should catch the SIGINT signal and flush the output stream before exiting.
msglog – message output collection from system startup or background applications

/dev/msglog

Output from system startup ("rc") scripts is directed to /dev/msglog, which dispatches it appropriately.

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>SUNWcsr</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

In the current version of Solaris, /dev/msglog is an alias for /dev/sysmsg. In future versions of Solaris, writes to /dev/msglog may be directed into a more general logging mechanism such as syslogd(1M).

syslog(3C) provides a more general logging mechanism than /dev/msglog and should be used in preference to /dev/msglog whenever possible.
**msm(7D)**

**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>IA</td>
</tr>
</tbody>
</table>

**SEE ALSO**
attributes(5)
### NAME
mt – tape interface

### DESCRIPTION
The files `rmt/*` refer to tape controllers and associated tape drives.

The `labelit(1M)` command requires these magnetic tape file names to work correctly with the tape controllers. No other tape controller commands require these file names.

### FILES
`/dev/rmt/*`

### SEE ALSO
`labelit(1M)`
mtio(7I)

NAME  mtio – general magnetic tape interface

SYNOPSIS  
```c
#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.

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.
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.

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 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(3AIO) and aiowrite(3AIO) functions.

To enable persistent error handling, the ioctl MTIOCPERSISTENT must be issued. If this ioctl succeeds, then persistent error handling is enabled and changes the current error behavior. This ioctl will fail if the device driver does not support persistent error handling.

With persistent error handling enabled, all tape operations after an exception or error will return with the same error as the first command that failed; the operations will not be executed. An exception is some event that might stop normal tape operations, such as an End Of File (EOF) mark or an End Of Tape (EOT) mark. An example of an error is a media error. The MTIOCLRERR ioctl must be issued to allow normal tape operations to continue and to clear the error.

Disabling persistent error handling returns the error behavior to normal SVR4 error handling, and will not occur until all outstanding operations are completed. Applications should wait for all outstanding operations to complete before disabling persistent error handling. Closing the device will also disable persistent error handling and clear any errors or exceptions.

The Read Operation and Write Operation subsections contain more pertinent information regarding persistent error handling.

The read(2) function reads the next record on the tape. The record size is passed back as the number of bytes read, provided it is not greater than the number requested. When a tape mark or end of data is read, a zero byte count is returned; all successive reads after the zero read will return an error and errno will be set to EIO. To move to the next file, an MTFSF ioctl can be issued before or after the read causing the error. This error handling behavior is different from the older BSD behavior, where another read will fetch the first record of the next tape file. If the BSD behavior is required, device names containing the letter b (for BSD behavior) in the final component should be used. If persistent error handling was enabled with either the BSD or SVR4 tape device behavior, all operations after this read error will return EIO errors until the MTIOCLRERR ioctl is issued. An MTFSF ioctl can then 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.

The write(2) function writes the next record on the tape. The record has the same length as the given buffer.

Writing is allowed on 1/4" tape at either the beginning of tape or after the last written file on the tape. With the Exabyte 8200, data may be appended only at the beginning of tape, before a filemark, or after the last written file on the tape.

Writing is not so restricted on 1/2", 4mm, and the other 8mm cartridge tape drives. Care should be used when appending files onto 1/2" reel tape devices, since an extra file mark is appended after the last file to mark the EOM. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the EOM ioctl is provided. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

Fixed-length I/O tape devices require the number of bytes written to be a multiple of the physical record size. For example, 1/4" cartridge tape devices only write multiples of 512 bytes.

Fixed-length I/O tape devices write multiple records if the blocking factor is greater than 64,512 bytes (minphys limit). These multiple writes are limited to 64,512 bytes. For example, if a write request is issued for 65,536 bytes using a 1/4" cartridge tape, two writes are issued; the first for 64,512 bytes and the second for 1024 bytes.

Most tape devices which support variable-length I/O operations may write a range of 1 to 65,535 bytes. If the record size exceeds 65,535 bytes, the driver writes multiple records to satisfy the request. These multiple records are limited to 65,534 bytes. As an example, if a write request for 65,540 bytes is issued, two records are written; one for
65,534 bytes followed by another record for 6 bytes. Newer variable-length tape drivers may relax the above limitation and allow applications to write record sizes larger than 65,534. Refer to the specific tape driver man page for details.

When logical EOT is encountered during a write, that write operation completes and the number of bytes successfully transferred is returned (note that a ‘short write’ may have occurred and not all the requested bytes would have been transferred. The actual amount of data written will depend on the type of device being used). The next write will return a zero byte count. A third write will successfully transfer some bytes (as indicated by the returned byte count, which again could be a short write); the fourth will transfer zero bytes, and so on, until the physical EOT is reached and all writes will fail with EIO.

When logical EOT is encountered with persistent error handling enabled, the current write may complete or be a short write. The next write will return a zero byte count. At this point an application should act appropriately for end of tape cleanup or issue yet another write, which will return the error ENOSPC. After clearing the exception with MTIOCLRERR, the next write will succeed (possibly short), followed by another zero byte write count, and then another ENOSPC error.

Allowing writes after LEOT has been encountered enables the flushing of buffers. However, it is strongly recommended to terminate the writing and close the file as soon as possible.

Seeks are ignored in tape I/O.

**Close Operation**

Magnetic tapes are rewound when closed, except when the “no-rewind” devices have been specified. The names of no-rewind device files use the letter n as the end of the final component. The no-rewind version of /dev/rmt/0l is /dev/rmt/0ln. In case of error for a no-rewind device, the next open rewinds the device.

If the driver was opened for reading and a no-rewind device has been specified, the close advances the tape past the next filemark (unless the current file position is at EOM), leaving the tape correctly positioned to read the first record of the next file. However, if the tape is at the first record of a file it doesn’t advance again to the first record of the next file. These semantics are different from the older BSD behavior. If BSD behavior is required where no implicit space operation is executed on close, the non-rewind device name containing the letter b (for BSD behavior) in the final component should be specified.

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device was specified, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning ioctl, like rewind, is issued after writing, a file mark is written before repositioning the tape.

All buffers are flushed on closing a tape device. Hence, it is strongly recommended that the application wait for all buffers to be flushed before closing the device. This can be done by writing a filemark via MTWEOF, even with a zero count.
Note that for 1/2” reel tape devices, two file marks are written to mark the EOM before rewinding or performing a file positioning ioctl. If the user wrote a file mark before closing a 1/2” reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly. If the non-rewinding device was specified, two file marks are written and the tape is left positioned between the two so that the second one is overwritten on a subsequent open(2) and write(2).

If no data was written and the driver was opened for WRITE-ONLY access, one or two file marks are written, thus creating a null file.

After closing the device, persistent error handling will be disabled and any error or exception will be cleared.

IOCTLs

Not all devices support all ioctl’s. The driver returns an ENOTTY error on unsupported ioctl’s.

The following structure definitions for magnetic tape ioctl commands are from <sys/mtio.h>.

The minor device byte structure is:

```
 15 7 6 5 4 3 2 1 0
```

<table>
<thead>
<tr>
<th>Bits 7-15</th>
<th>BSD</th>
<th>Reserved</th>
<th>Density</th>
<th>Density</th>
<th>No rewind</th>
<th>Unit #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior</td>
<td>Select</td>
<td>Select on Close</td>
<td>Bits 0-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/* Layout of minor device byte: */
#define MTUNIT(dev) (((minor(dev) & 0xff80) >> 5) + (minor(dev) & 0x3))
#define MT_NOREWIND (1 <<2)
#define MT_DENSITY_MASK (3 <<3)
#define MT_DENSITY1 (0 <<3) /* Lowest density/format */
#define MT_DENSITY2 (1 <<3)
#define MT_DENSITY3 (2 <<3)
#define MT_DENSITY4 (3 <<3) /* Highest density/format */
#define MTMINOR(unit) (((unit & 0x7fc) << 5) + (unit & 0x3))
#define MT_BSD (1 <<6) /* BSD behavior on close */

/* Structure for MTIOCTOP - magnetic tape operation command */
struct mtop {
  short mt_op;  /* operation */
  daddr_t mt_count;  /* number of operations */
};

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
MTREW rewind
MTOFFL rewind and take the drive off-line
MTNOP no operation, sets status only
MTRE TEN retension the tape (cartridge tape only)
MTERASE erase the entire tape and rewind
MTEOM position to EOM
MTNBSF backward space file to beginning of file
MTSRSZ set record size
MTGRSZ get record size
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 */
    short mt_dsreg; /* drive status register */
    short mt_erreg; /* error register */
    daddr_t mt_resid; /* residual count */
    daddr_t mt_fileno; /* file number of current position */
    daddr_t mt_blkno; /* block number of current position */
    ushort_t mt_flags;
    short mt_bf; /* optimum blocking factor */
};

/* structure for MTIOCGETDRIVETYPE — get tape config data command */
struct mtdrivetype_request {
    int size;
    struct mtdrivetype *mtdtp;
};

struct mtdrivetype {
    char name[64]; /* Name, for debug */
    char vid[25]; /* Vendor id and product id */
    char type; /* Drive type for driver */
    int bsize; /* Block size */
    int options; /* Drive options */
    int max_rretries; /* Max read retries */
    int max_wretries; /* Max write retries */
    uchar_t densities[MT_NDENSITIES]; /* density codes, low->hi */
    uchar_t default_density; /* Default density chosen */
    uchar_t speeds[MT_NSPEEDS]; /* speed codes, low->hi */
};
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 offlineing 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 `MTRETN` 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.
**Persistent Error Handling IOCTLs and Asynchronous Tape Operations**

- **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 **mtop** struct is used to pass the record size to/from the st driver. A value of 0 indicates variable record size. The **MTSRSZ** ioctl makes a variable-length tape device behave like a fixed-length tape device. Refer to the specific tape driver man page for details.

- **MTLOAD**: Loads the next tape cartridge into the tape drive. This is generally only used with stacker and tower type tape drives which handle multiple tapes per tape drive. A tape device without a tape inserted can be opened with the **O_NDELAY** flag, in order to execute this operation.

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**).

- **MTIOC Persistent** enables/disables persistent error handling
- **MTIOC Persistent Status** queries for persistent error handling
- **MTIOC CLR ERR** clears persistent error handling
- **MTIOC GUARANTEED ORDER** checks whether driver guarantees order of I/O’s
The `MTIOCPERSISTENT` ioctl enables or disables persistent error handling. It takes as an argument a pointer to an integer that turns it either on or off. If the ioctl succeeds, the desired operation was successful. It will wait for all outstanding I/Os 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.

### 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 ioctl’s will wait for all outstanding commands to complete before they are executed.
### IOCTLs for Multi-initiator Configurations

<table>
<thead>
<tr>
<th>IOCTL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTIOCRESERVE</td>
<td>reserve the tape drive</td>
</tr>
<tr>
<td>MTIOCRELEASE</td>
<td>revert back to the default behavior of reserve on open/release on close</td>
</tr>
<tr>
<td>MTIOCFORCERESERVE</td>
<td>reserve the tape unit by breaking reservation held by another host</td>
</tr>
</tbody>
</table>

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 unit. This ioctl can be executed only with super-user privileges. It is recommended to open the tape device in `O_NDELAY` mode when this ioctl needs to be executed, otherwise the open will fail if another host indeed has it reserved. For example,

```c
ioctl(fd, MTIOCFORCERESERVE);
```

### IOCTLs for Handling Tape Configuration Options

<table>
<thead>
<tr>
<th>IOCTL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTIOCSHORTFMK</td>
<td>enables/disable support for writing short filemarks. This is specific to Exabyte drives.</td>
</tr>
<tr>
<td>MTIOCREADIGNOREILI</td>
<td>enables/disable suppress incorrect length indicator support during reads</td>
</tr>
<tr>
<td>MTIOCREADIGNOREEEOFs</td>
<td>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</td>
</tr>
</tbody>
</table>

The `MTIOCSHORTFMK` ioctl enables or disables support for short filemarks. This ioctl is only applicable to Exabyte drives which support short filemarks. As an argument, it takes a pointer to an integer. If 0 (zero) is the specified integer, then long filemarks will be written. If 1 is the specified integer, then short filemarks will be written. The specified tape behavior will be in effect until the device is closed.

For example:

```c
int on = 1;
int off = 0;
/* enable short filemarks */
```
ioctl(fd, MTIOSHORTFMK, &on);
/* disable short filemarks */
ioctl(fd, MTIOSHORTFMK, &off);

Tape drives which do not support short filemarks will return an \texttt{errno} of \texttt{ENOTTY}.

The \texttt{MTIOCREADIGNOREILI} ioctl enables or disables the suppress incorrect length indicator (SILI) support during reads. As an argument, it takes a pointer to an integer. If 0 (zero) is the specified integer, SILI will not be used during reads and incorrect length indicator will not be supressed. If 1 is the specified integer, SILI will be used during reads and incorrect length indicator will be supressed. The specified tape behavior will be in effect until the device is closed.

For example:

```c
int on = 1;
int off = 0;
ioctl(fd, MTIOCREADIGNOREILI, &on);
ioctl(fd, MTIOCREADIGNOREILI, &off);
```

The \texttt{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, MTIOCREADIGNOREEOFS, &on);
ioctl(fd, MTIOCREADIGNOREEOFS, &off);
```

Tape drives other than 1/2" reel tapes will return an \texttt{errno} of \texttt{ENOTTY}.

**EXAMPLES**

**EXAMPLE 1** Tape Positioning and Tape Drives

Suppose you have written three files to the non-rewinding 1/2" tape device, /dev/rmt/0ln, and that you want to go back and \texttt{dd} (1M) the second file off the tape. The commands to do this are:

```sh
mt -F /dev/rmt/0lbn bsf 3
mt -F /dev/rmt/0lbn fsf 1
dd if=/dev/rmt/0ln
```

To accomplish the same tape positioning in a C program, followed by a get status ioctl:
EXAMPLE 1 Tape Positioning and Tape Drives (Continued)

```c
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

```c
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:

```c
struct mtdrivetype mtdt;
struct mtdrivetype_request mtreq;
mtreq.size = sizeof(struct mtdrivetype);
mtreq.mtdtp = &mtdt;
ioctl(fd, MTIOCGETDRIVETYPE, &mtreq);
```

FILES

```
/dev/rmt/<unit number><density>[<BSD behavior>][<no rewind>]
```

Where `density` can be l, m, h, u/c (low, medium, high, ultra/compressed, respectively), the `BSD behavior` option is b, and the `no rewind` option is n.

For example, `/dev/rmt/0hbn` specifies unit 0, high density, BSD behavior and no rewind.

SEE ALSO

mt(1), tar(1), dd(1M), open(2), read(2), write(2), aioread(3AIO),
aiowrite(3AIO), ar(3HEAD), st(7D)

1/4 Inch Tape Drive Tutorial
ncrs(7D)

NAME
ncrs – ncrs SCSI host bus adapter driver

SYNOPSIS
scsi@unit-address

DESCRIPTION
The ncrs host bus adapter driver is a SCSA-compliant nexus driver that supports the LSI Logic (formerly Symbios Logic or NRC) 53C810, 53C810A, 53C815, 53C820, 53C825, 53C825A, 53C860, 53C875, 53C875J, 53C876, and 53C895 SCSI (Small Computer Systems Interface) chips.

The ncrs driver supports standard functions provided by the SCSA interface, including tagged and untagged queuing, Wide/Fast/Ultra/Ultra2 SCSI, and auto request sense. The ncrs driver does not support linked commands.

Preconfiguration Information

- The NCR BIOS and the Solaris fdisk program may be incompatible. To prevent conflicts, you should create an entry in the FDISK partition table using the DOS version of FDISK (or equivalent utility) before installing the Solaris software. To ensure your system will reboot following Solaris installation, create a DOS partition at least 1-cylinder in size that starts at cylinder 0.

- Add-in cards containing 53C815, 53C820, 53C825, or 53C825A controllers must be used in bus-mastering PCI slots. PCI slots on dual PCI slot motherboards are generally bus-master capable. However, motherboards that contain three or more PCI slots, or motherboards that feature several embedded PCI controllers may contain PCI slots that are not bus-master capable.

- PCI motherboards that feature Symbios Logic SDMS BIOS and an embedded 53C810 or 53C810A controller may not be compatible with 53C82x add-in cards equipped with Symbios Logic SDMS BIOS. To prevent conflicts, it may be necessary to upgrade the motherboard BIOS, the add-in card, or both.

- Early PCI systems that are equipped with a 53C810 motherboard chip may contain unconnected interrupt pins. These systems cannot be used with Solaris software.

- Wide-to-narrow target connections are not supported by Solaris software; as a result, you should not attempt to connect wide targets to narrow connectors on any of the supported devices.

- If your adapter supports the Symbios Logic SCSI configuration utility, the value of the hosts SCSI ID (found under the Adapter Setup menu) must be set to 7. (You can access the Symbios Logic SCSI configuration utility using Control-C.)

- If you experience problems with old target devices, add the following to the /kernel/drv/ncrs.conf file:

  targetN-scsi-options = 0x0;
  where N is the ID of the failing target.

- If you are using a Conner 1080S narrow SCSI drive, the system may display the following warnings:

  WARNING: /pci@0,0/pci1000, f@d (ncrs0):
  invalid reselection (0,0)
WARNING: /pci@0,0/pci1000,f@d/sd@0,0 (sd0); SCSI transport failed: ‘reset: retrying command’

To suppress these warnings, disable tagged queuing in the ncrs.conf file.

- Pentium motherboards (Intel NX chipset) using P90 or slower processors may cause the ncrs driver to hang. If this occurs, the following messages are displayed on the console:

warning /pci@0,0/pci1000,3@6(ncrs0)
Unexpected DMA state:active dstat=c0<DMA-FIFO-empty, master-data-parity-error>

This is an unrecoverable state and the system will not install using the ncrs driver.

- The ncrs driver supports the 53C875 chipset Revision 4, or later versions only. Earlier, pre-release versions of the chip are not supported.

- On rare occasions, use of an SDT7000/SDT9000 tape drive may result in the following message being displayed on the console:

Unexpected DMA state: ACTIVE. dstat=81<DMA-FIFO-empty, illegal-instruction>

After the above message is displayed, the system and tape drive will recover and remain usable.

The ncrs host bus adapter driver is configured by defining the properties found in ncrs.conf. Properties in the ncrs.conf file that can be modified by the user include: scsi-options, target<n>-scsi-options, scsi-reset-delay, scsi-tag-age-limit, scsi-watchdog-tick, scsi-initiator-id, and ncrs-iomap. Properties in the ncrs.conf file override global SCSI settings.

The property target<n>-scsi-options overrides the scsi-options property value for target<n>, where <n> can vary from decimal 0 to 15. The ncrs driver supports the following SCSI options: SCSI_OPTIONS_DR(0x8), SCSI_OPTIONS_SYNC(0x20), SCSI_OPTIONS_TAG(0x80), SCSI_OPTIONS_FAST(0x100), SCSI_OPTIONS_WIDE(0x200), SCSI_OPTIONS_FAST20(0x400), and SCSI_OPTIONS_FAST40(0x800).

After periodic interval scsi-watchdog-tick, the ncrs driver searches through all current and disconnected commands for timeouts.

The scsi-tag-age-limit property represents the number of times that the ncrs driver attempts to allocate a tag ID that is currently in use after going through all tag IDs in a circular fashion. When encountering the same tag ID used scsi-tag-age-limit times, no additional commands are submitted to the target until all outstanding commands complete or timeout.

The ncrs-iomap property enables the driver to utilize IO mapping (rather than memory mapping) of registers.

Refer to scsi_hba_attach(9F) for details.
ncrs(7D)

EXAMPLES

EXAMPLE 1 A sample ncrs configuration file

Create a file called /kernel/drv/ncrs.conf, then add the following line:

```
scsi-options=0x78;
```

The above example disables tagged queuing, Fast/Ultra SCSI, and wide mode for all ncrs instances.

The following example disables an option for one specific ncrs device. See driver.conf(4) and pci(4) for more details.

```
name="ncrs" parent="/pci@1f,4000"
  unit-address="3"
  target1-scsi-options=0x58
  scsi-options=0x178 scsi-initiator-id=6;
```

In the example, the default initiator ID in OBP is 7; the change to ID 6 will occur at attach time. The scsi-options property is set for target 1 to 0x58 and all other targets set to 0x178. Note that it may be preferable to change the initiator ID in OBP.

The physical path name of the parent can be determined using the /devices tree or by following the link of the logical device name:

```
# ls -l /dev/rdsk/c0t0d0s0
lrwxrwxrwx 1 root root 45 May 16 10:08 /dev/rdsk/c0t0d0s0 ->
  ../.. /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

In the example above, the parent is /pci@1f,4000 and the unit-address is the number bound to the scsi@3 node.

To set scsi-options more specifically per target, do the following:

```
  target1-scsi-options=0x78;
  device-type-scsi-options-list =
"SEAGATE ST32550W", "seagate-scsi-options" ;
  seagate-scsi-options = 0x58;
  scsi-options=0x3f8;
```

With the exception of one specific disk type that has scsi-options set to 0x58, the example above sets scsi-options for target 1 to 0x78 and all other targets to 0x3f8.

The scsi-options properties that are specified per target ID have the highest precedence, followed by scsi-options per device type. Global scsi-options (for all ncrs instances) per bus have the lowest precedence.

To turn on IO mapping for all ncrs cards in the system, do the following:

```
ncrs-iomap=1;
```

The above action will noticeably slow the performance of the driver. You must reboot the system for the specified scsi-options to take effect.
To enable some driver features, the target driver must set capabilities in the ncrs driver. The following capabilities can be queried and modified by the target driver:
synchronous, tagged-qing, wide-xfer, auto-rqsense, qfull-retries, and qfull-retry-interval. All other capabilities are query only.

The tagged-qing, auto-rqsense, wide-xfer, disconnect, and Ultra/ Ultra2 synchronous capabilities are enabled by default, and can be assigned binary (0 or 1) values only. The default value for qfull-retries is 10, while the default value for qfull-retry-interval is 100. The qfull-retries capability is a uchar_t (0 to 255), while qfull-retry-interval is a ushort_t (0 to 65535).

If a conflict exists between the value of scsi-options and a capability, the value set in scsi-options prevails. Only whom != 0 is supported in the scsi_ifsetcap(9F) call. Refer to scsi_ifsetcap(9F) and scsi_ifgetcap(9F) for details.

The ncrs host bus adapter driver also supports hotplugging of targets using the cfgadm tool. Hotplug operations on the SCSI bus that hosts the root partition should not be performed. See the cfgadm(1M) man page for more information.

FILES

/kernel/drv/ncrs ELF kernel module
/kernel/drv/ncrs.conf Optional configuration file

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
</table>

SEE ALSO

prtconf(1M), driver.conf(4), pci(4), attributes(5), scsi_abort(9F), scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F), scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

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

Symbios Logic Inc., SYM53C895 PCI-Ultra2 SCSI I/O Processor With LVDlink

Symbios Logic Inc., SYM53C875 PCI-SCSI I/O Processor With Fast-20

Symbios Logic Inc., SYM53C825A PCI-SCSI I/O Processor

Symbios Logic Inc., SYM53C810A PCI-SCSI I/O Processor
The messages described below are logged and may also appear on the system console.

Device is using a hilevel intr

The device was configured with an interrupt level that cannot be used with this ncrs driver. Check the PCI device.

map setup failed

The driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

glm_script_alloc failed

The driver was unable to load the SCRIPTS for the SCSI processor; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

cannot map configuration space

The driver was unable to map in the configuration registers. Check for bad hardware. SCSI devices will be inaccessible.

attach failed

The driver was unable to attach; usually preceded by another warning that indicates why attach failed. These can be considered hardware failures.

SCSI bus DATA IN phase parity error

The driver detected parity errors on the SCSI bus.

SCSI bus MESSAGE IN phase parity error

The driver detected parity errors on the SCSI bus.

SCSI bus STATUS phase parity error

The driver detected parity errors on the SCSI bus.

Unexpected bus free

Target disconnected from the bus without notice. Check for bad hardware.

Disconnected command timeout for Target <id>.<lun>

A timeout occurred while target id/lun was disconnected. This is usually a target firmware problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.

Disconnected tagged cmd(s) (<n>) timeout for Target <id>.<lun>

A timeout occurred while target id/lun was disconnected. This is usually a target firmware problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.
Connected command timeout for Target <id>.<lun>

This is usually a SCSI bus problem. Check cables and termination.

Target <id> reducing sync. transfer rate

A data transfer hang or DATA-IN phase parity error was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> reverting to async. mode

A second data transfer hang was detected for this target. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> disabled wide SCSI mode

A second data phase hang was detected for this target. The driver attempts to eliminate this problem by disabling wide SCSI mode.

auto request sense failed

An attempt to start an auto request packet failed. Another auto request packet may already be in transport.

invalid reselection (<id>.<lun>)

A reselection failed; target accepted abort or reset, but still tries to reconnect. Check for bad hardware.

invalid intcode

The SCRIPTS processor generated an invalid SCRIPTS interrupt. Check for bad hardware.

NOTES

The ncrs hardware (53C875) supports Wide, Fast, and Ultra SCSI mode. The maximum SCSI bandwidth is 40 MB/sec.

The ncrs hardware (53C895) supports Wide, Fast, Ultra and Ultra2 SCSI mode using a LVD bus. The maximum SCSI bandwidth is 80 MB/second.

The ncrs driver exports properties indicating the negotiated transfer speed per target (target<n>-sync-speed), whether wide bus is supported (target<n>-wide) for that particular target (target<n>-scsi-options), and whether tagged queuing has been enabled (target<n>-TQ). The sync-speed property value indicates the data transfer rate in KB/sec. The target<n>-TQ and the target<n>-wide property have value 1 (to indicate that the corresponding capability is enabled for that target), or 0 (to indicate that the capability is disabled for that target). See prtconf(1M) (verbose option) for details on viewing the ncrs properties.

scsi, instance #0

Driver properties:

name <target6-TQ> length <4>
value <0x00000000>.
name <target6-wide> length <4>
  value <0x00000000>.
name <target6-sync-speed> length <4>
  value <0x00002710>.
name <target1-TQ> length <4>
  value <0x00000001>.
name <target1-wide> length <4>
  value <0x00000000>.
name <target1-sync-speed> length <4>
  value <0x00002710>.
name <target0-TQ> length <4>
  value <0x00000001>.
name <target0-wide> length <4>
  value <0x00000001>.
name <target0-sync-speed> length <4>
  value <0x00009c40>.
name <scsi-options> length <4>
  value <0x000007f8>.
name <scsi-watchdog-tick> length <4>
  value <0x0000000a>.
name <scsi-tag-age-limit> length <4>
  value <0x00000002>.
name <scsi-reset-delay> length <4>
  value <0x00000bb8>.
name <latency-timer> length <4>
  value <0x00000088>.
name <cache-line-size> length <4>
  value <0x00000010>.
null – the null file, also called the null device

/dev/null

Data written on the null special file, /dev/null, is discarded.

Reads from a null special file always return 0 bytes.

/files

/dev/null
### NAME
ocf_escr1 – Sun external serial smart card terminal driver

### DESCRIPTION
The `ocf_escr1` driver is an OpenCard Framework (OCF)-compliant card terminal driver for the Sun external serial smart card reader.

The `ocf_escr1` driver is part of the OCF framework stack and is started by the OCF startup script. The Sun serial smart card reader requires a host serial port and is accessed through the character-special devices. The reader is powered from the keyboard port.

### FILES
- `/usr/share/lib/smartcard/scmrsr3.jar`
  - Java-archived driver class files
- `/dev/cua/x`
  - Asynchronous serial line using port x

### SEE ALSO
`ports(1M), smartcard(1M), smartcard(5)`
### NAME
ocf_ibutton – iButton Smart Card terminal driver

### DESCRIPTION
The ocf_ibutton smart card terminal driver is an OpenCard Framework (OCF)-compliant terminal driver for the Dallas Semiconductor iButton reader.

### APPLICATION PROGRAMMING INTERFACE
The ocf_ibutton smart card terminal driver is part of the OCF framework stack and is started by the OCF startup script. The iButton reader requires a host serial port and is accessed through the character-special devices.

### FILES
- `/usr/share/lib/smartcard/ibutton.jar`
  - Java-archived driver class files
- `/dev/cua/x`
  - Asynchronous serial line using port x

### SEE ALSO
- `ports(1M)`, `smartcard(1M)`, `smartcard(5)`
<table>
<thead>
<tr>
<th>NAME</th>
<th>ocf_iscr1 – I2C smart card card terminal driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>The ocf_iscr1 I2C smart card card terminal driver is an OpenCard Framework (OCF)-compliant terminal driver for SCM Microsystems Smart Transporter chips that feature the I2C bus interface.</td>
</tr>
<tr>
<td>APPLICATION</td>
<td>The ocf_iscr1 I2C driver is part of the OCF framework stack and is started by the OCF server daemon. The smart card reader requires the /platform/sun4u/kernel/drv/sparcv9/scmi2c Solaris hardware device driver to be installed and present to work. The smart card reader driver also requires device node /dev/scmi2cn, where n is the nth SCM I2C card terminal reader installed.</td>
</tr>
<tr>
<td>PROGRAMMING</td>
<td>FILES</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>/usr/share/lib/smartcard/scmiscr.jar</td>
</tr>
<tr>
<td></td>
<td>Java-archived driver class files</td>
</tr>
<tr>
<td></td>
<td>/usr/share/lib/smartcard/smartos.jar</td>
</tr>
<tr>
<td></td>
<td>SCM Microsystems SmartOS Java-archived driver class file</td>
</tr>
<tr>
<td></td>
<td>/dev/scmi2cn</td>
</tr>
<tr>
<td></td>
<td>SCM Microsystems Smart Transporter chip device node</td>
</tr>
<tr>
<td></td>
<td>/platform/sun4u/kernel/drv/sparcv9/scmi2c</td>
</tr>
<tr>
<td></td>
<td>SCM Microsystems Smart Transporter chip kernel module</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>smartcard(1M), smartcard(5)</td>
</tr>
</tbody>
</table>
NAME | ohci – OpenHCI host controller driver
SYNOPSIS | usba@unit-address
DESCRIPTION | The ohci driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the Open Host Controller Interface Specification 1.0a, an industry standard developed by Compaq, Microsoft, and National Semiconductor.

The ohci driver supports bulk, interrupt, control and isochronous transfers. It supports the nexus device control interface.

FILES | /kernel/drv/ohci
32 bit ELF kernel module
/kernel/drv/sparcv9/ohci
64 bit 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>PCI-based SPARC systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO | hid(7D), hubd(7D), uhci(7D), scsa2usb(7D), hid(7D), usba(7D)

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1
Open Host Controller Interface Specification for USB 1.0a
System Administration Guide: Basic Administration

DIAGNOSTICS | All host controller errors are passed to the client drivers. Root errors are documented in hubd(7D).

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

WARNING: <device path> <ohci<instance number>>: Error message...

Unrecoverable USB Hardware Error.

There was an unrecoverable USB hardware error reported by the OHCI Controller. Please reboot the system. If this problem persists, contact your system vendor.

No SOF interrupts.

The USB hardware error is not generating Start Of Frame interrupts. Please reboot the system. If this problem persists, contact your system vendor.
openprom(7D)

NAME  
openprom – PROM monitor configuration interface

SYNOPSIS  
#include <sys/fcntl.h>
#include <sys/types.h>
#include <sys/openpromio.h>
open(*/dev/openprom*, mode);

DESCRIPTION  
The internal encoding of the configuration information stored in EEPROM or NVRAM
varies from model to model, and on some systems the encoding is “hidden” by the
firmware. The openprom driver provides a consistent interface that allows a user or
program to inspect and modify that configuration, using ioctl(2) requests. These
requests are defined in <sys/openpromio.h>:

```
struct openpromio {
    uint_t oprom_size; /* real size of following data */
    union {
        char b[1]; /* NB: Adjacent, Null terminated */
        int i;
    } opio_u;
};
#define oprom_array opio_u.b /* property name/value array */
#define oprom_node opio_u.i /* nodeid from navigation config-ops */
#define oprom_len opio_u.i /* property len from OPROMGETPROPLEN */
#define OPROMMAXPARAM 32768 /* max size of array (advisory) */
```

For all ioctl(2) requests, the third parameter is a pointer to a struct openpromio.
All property names and values are null-terminated strings; the value of a numeric
option is its ASCII representation.

IOCTLS  

OPROMGETOPT  
This ioctl takes the null-terminated name of a property
in the oprom_array and returns its null-terminated value
(overlying its name). oprom_size should be set to the
size of oprom_array; on return it will contain the size of
the returned value. If the named property does not
exist, or if there is not enough space to hold its value,
then oprom_size will be set to zero. See BUGS below.

OPROMSETOPT  
This ioctl takes two adjacent strings in oprom_array; the
null-terminated property name followed by the
null-terminated value.

OPROMSETOPT2  
This ioctl is similar to OPROMSETOPT, except that it
uses the difference between the actual user array size
and the length of the property name plus its null
terminator.

OPROMNXTOPT  
This ioctl is used to retrieve properties sequentially.
The null-terminated name of a property is placed into
oprom_array and on return it is replaced with the
null-terminated name of the next property in the
sequence, with oprom_size set to its length. A null string
on input means return the name of the first property; an oprom_size of zero on output means there are no more properties.

OPROMNXT
OPROMCHILD
OPROMGETPROP
OPROMNXTPROP
These ioctls provide an interface to the raw config_ops operations in the PROM monitor. One can use them to traverse the system device tree; see prtconf(1M).

OPROMGETPROPLEN
This ioctl provides an interface to the property length raw config op. It takes the name of a property in the buffer, and returns an integer in the buffer. It returns the integer -1 if the property does not exist; 0 if the property exists, but has no value (a boolean property); or a positive integer which is the length of the property as reported by the PROM monitor. See BUGS below.

OPROMGETVERSION
This ioctl returns an arbitrary and platform-dependent NULL-terminated string in oprom_array, representing the underlying version of the firmware.

ERRORS
EAGAIN There are too many opens of the /dev/openprom device.
EFAULT A bad address has been passed to an ioctl(2) routine.
EINVAL The size value was invalid, or (for OPROMSETOPT) the property does not exist, or 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 oprom_array Data Allocation and Reuse

The following example shows how the oprom_array is allocated and reused for data returned by the driver.

/*
 * 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 <unistd.h>
EXAMPLE 1 oprom_array Data Allocation and Reuse (Continued)

```c
#include <sys/openpromio.h>
#define min(a, b) (a < b ? a : b)
#define max(a, b) (a > b ? a : b)
#define MAXNAMESZ 32    /* Maximum property *name* size */
#define BUFSZ 1024      /* A handy default buffer size */
#define MAXVALSZ (BUFSZ - sizeof (int))
static char *promdev = "/dev/openprom";
/*
 * Allocate an openpromio structure big enough to contain
 * a bufsize'd oprom_array. Zero out the structure and
 * set the oprom_size field to bufsize.
 */
static struct openpromio *
op_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)
{
    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);
}
```

```c
int
main(void)
{
    struct openpromio *opp;
    int fd, proplen;
```
EXAMPLE 1 oprom_array Data Allocation and Reuse

(Continued)

```c
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 = max(buflen, strlen("name") + 1);
opp = opp_zalloc(buflen);
(void) strcpy(opp->oprom_array, "name");
if (ioctl(fd, OPROMGETPROP, opp) < 0) {
    perror("OPROMGETPROP");
    exit(1);
}
if (opp->oprom_size != 0)
    printf("Platform name <%s> property len <%d>\n", opp->oprom_array, proplen);
opp_free(opp);
/*
* Allocate an openpromio structure assumed to be
* big enough to get the 'prom version string'.
* Get and print the prom version.
*/
```

EXAMPLE 1  oprom_array  Data Allocation and Reuse  (Continued)

    */
    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);
}

/dev/openprom  PROM monitor configuration interface

SEE ALSO  eeeprom(1M), monitor(1M), prtconf(1M), ioctl(2), mem(7D)

BUGS  There should be separate return values for non-existent properties as opposed to not
       enough space for the value.

       An attempt to set a property to an illegal value results in the PROM setting it to some
       legal value, with no error being returned. An OPROMGETOPT should be performed
       after an OPROMSETOPT to verify that the set worked.

       Some PROMS lie about the property length of some string properties, omitting the
       NULL terminator from the property length. The openprom driver attempts to
       transparently compensate for these bugs when returning property values by NULL
       terminating an extra character in the user buffer if space is available in the user buffer.
       This extra character is excluded from the oprom_size field returned from
       OPROMGETPROP and OPROMGETOPT and excluded in the oprom_len field returned from
       OPROMGETPROPLEN but is returned in the user buffer from the calls that return data, if
       the user buffer is allocated at least one byte larger than the property length.
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>IA</td>
<td>/devices/isa/pcic@1,3e0 /disk@0:a</td>
<td>/dev/dsk/c1d0s0</td>
</tr>
<tr>
<td>SPARC</td>
<td>/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:

<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)
The `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 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 IA platform. The 3C589B and 3C589C are supported on the SPARC platform.

```
FILES
/kernel/drv/pcelx pcelx driver
/dev/pcelx DLPI Style 2 device
/dev/pcelxn DLPI Style 1 device where: n is the PCMCIA physical socket number.
SEE ALSO
pcmcia(4)
```
pcfs – 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

pcfs is a file system type that allows users direct access to files on DOS formatted disks from within the SunOS operating system. Once mounted, pcfs 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 pcfs file system contained on the block special file identified by spec is mounted on the directory identified by dir. spec and dir are pointers to pathnames. mflag specifies the mount options. The MS_DATA bit in mflag must be set. Mounting a pcfs file system requires a pointer to a structure containing mount flags and local timezone information, *pc_argp:

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 members of this structure is described in ctime(3C). flags can contain the PCFS_MNT_FOLDCASE flag. Fold names read from the file system to lowercase.

Mounting File Systems

Use the following command to mount pcfs from diskette:

    mount -F pcfs device-special directory-name

You can use:

    mount directory-name

if the following line is in your /etc/vfstab file:

    device-special - directory-name pcfs - no rw

Use the following command to mount pcfs from non-diskette media:

    mount -F pcfs device-special:logical-drive directory-name

You can use:

    mount directory-name

if the following line is in your /etc/vfstab file:

    device-special:logical_drive - directory-name pcfs - no rw
device-special specifies the special block device file for the diskette (/dev/disketteN) or the entire hard disk (/dev/dsk/cNtNdNp0 for a SCSI disk, and /dev/dsk/cNdNp0 for IDE disks) or the PCMCIA pseudo-floppy memory card (/dev/dsk/cNtNdNsN).

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, to mount the Primary DOS partition from a SCSI hard disk, use:

```
mount -F pcfs /dev/dsk/cNtNdNp0:c /pcfs/c
```

To mount the first logical drive in the Extended DOS partition from an IDE hard disk, use:

```
mount -F pcfs /dev/dsk/cNdNp0:d /pcfs/d
```

To mount a DOS diskette in the first floppy drive, 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/cNtNdNsN /pcfs
```

Conventions

Files and directories created through pcfs have to comply with either the DOS short file name convention or the long file name convention introduced with Windows 95. The DOS short file name convention is of the form filename[.ext], where filename generally consists of from one to eight upper-case characters, while the optional ext consists of from one to three upper-case characters.

The long file name convention is much closer to Solaris file names. A long file name can consist of any any characters valid in a short file name, lower-case letters, non-leading spaces, the characters +, ,, = [ ] , any number of periods, and can be up to 255 characters long. Long file names have an associated short file name for systems that do not support long file names (such as earlier releases of Solaris). The short file name is not visible if the system recognizes long file names. pcfs generates a unique short name automatically when creating a long file name.

Given a long file name such as This is a really long filename.TXT, the short file name will generally be of the form THISIS~N.TXT, where N is a number. So, this...
When creating a file name, pcfs creates a short file name if it fits the DOS short file name format, otherwise it creates a long file name. This is because long file names take more directory space. In fact, since the root directory of a pcfs file system is fixed size, long file names in the root directory should be avoided if possible.

When displaying file names, pcfs shows them exactly as they are on the media (so short names show up as all uppercase, and long file names retain their case). The old behavior of pcfs was to fold all names to lowercase, which can be forced with the PCFS_MNT_FOLDCASE mount option. All file name searches within pcfs, however, are treated as if they were uppercase, so readme.txt and ReAdMe.TxtT refer to the same file.

To format a diskette or a PCMCIA pseudo-floppy memory card in DOS format in the SunOS system, use either the fdformat-d or the DOS FORMAT command.

On IA systems, hard drives may contain an fdisk partition reserved for the Solaris boot utilities. These partitions are special instances of pcfs. You can mount an IA boot partition with the command:

```
mount -F pcfs device-special:boot directory-name
```

or you can use:

```
mount directory-name
```

if the following line is in your /etc/vfstab file:

```
device-special:boot - directory-name pcfs - no rw
```

`device-special` specifies the special block device file for the entire hard disk (/dev/dsk/cNdNp0)

`directory-name` specifies the location where the file system is mounted.

All files on a boot partition are owned by super-user. Only the super-user may create, delete, or modify files on a boot partition.

**EXAMPLES**

**EXAMPLE 1 Sample Displays of File Names**

If you copy a file `financial.data` from a UNIX file system to pcfs, it will show up as `financial.data` in pcfs, but will probably show up as FINANC-1.DAT in systems that do not support long file names.

The following file names are not legal short file names, but are legal long file names:

```
test.sh.orig
```
EXAMPLE 1 Sample Displays of File Names (Continued)

data+
    .login

Other systems that do not support long file names may well see:

TESTSH-1.ORI
DATA-1
LOGIN-1

The short file name is generated from the initial characters of the long file name, so it is better to differentiate names in the first few characters. For example, these names:

WorkReport.February.Data
WorkReport.March.Data

result in these short names, which are not very distinguishable:

WORKRE-1.DAT
WORKRE-2.DAT
WORKRE-13.DAT

These names, however:

January.WorkReport.Data
February.WorkReport.Data
March.WorkReport.Data

result in the more descriptive short names:

JANUAR-1.DAT
FEBRUA-1.DAT
MARCHW-1.DAT

FILES
/usr/lib/fs/pcfs/mount   pcfs mount command
/usr/kernel/fs/pcfs     32-bit kernel module
/usr/kernel/fs/sparcv9/pcfs  64-bit kernel module

SEE ALSO
chgrp(1), chown(1), dos2unix(1), eject(1), fdformat(1), unix2dos(1),
volcheck(1), mount(1M), mount_pcfs(1M), vold(1M), ctime(3C), vfstab(4),
pcmem(7D)
Do not physically eject a DOS floppy while the device is still mounted as \texttt{pcfs}. If Volume Management is managing a device, use the \texttt{eject(1)} command before physically removing media.

When mounting \texttt{pcfs} on a hard disk, make sure the first block on that device contains a valid \texttt{fdisk} partition table.

Because \texttt{pcfs} has no provision for handling owner-IDs or group-IDs on files, \texttt{chown(1)} or \texttt{chgrp(1)} may generate various errors. This is a limitation of \texttt{pcfs}, but it should not cause problems other than error messages.

The following characters are the only ones allowed in \texttt{pcfs} short file names and extensions:

\begin{itemize}
  \item 0-9
  \item A-Z
  \item $@!\%()-{}<>'^_\|
\end{itemize}

SunOS and DOS use different character sets and have different requirements for the text file format. Use the \texttt{dos2unix(1)} and \texttt{unix2dos(1)} commands to convert files between them.

\texttt{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. Use \texttt{ufs} instead for local storage within the SunOS system.

Although long file names can contain spaces (just as in UNIX file names), some utilities may be confused by them.

This implementation of \texttt{pcfs} conforms to the behavior exhibited by Windows 95 version 4.00.950.

\texttt{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.

When listing or searching a directory, \texttt{pcfs} does not include files with the \texttt{hidden} or \texttt{system} bits set.
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:

- Intel 82365SL
- Cirrus Logic PD6710/PD6720/PD6722
- Vadem VG365/VG465/VG468/VG469
- Toshiba PCIC and ToPIC
- Ricoh RF5C366
- Texas Instruments PCI1130/PCI1131/PCI1031

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.

**Driver Configuration**

There is one driver configuration property defined in the `pcic.conf` file.

```plaintext
interrupt-priorities=11;
```

This property must be defined and must not be modified from the default value.

**FILES**

- `/kernel/drv/pcic`  pcic driver
- `/kernel/drv/pcic.conf`  pcic configuration file

**SEE ALSO**

pcmcia(4) and stp4020(7D)
pckt – STREAMS Packet Mode module

SYNOPSIS

```c
int ioctl(fd, I_PUSH, "pckt");
```

DESCRIPTION

pckt is a STREAMS module that may be used with a pseudo terminal to packetize certain messages. The pckt module should be pushed (see `I_PUSH` on `streamio(7I)`) onto the master side of a pseudo terminal.

Packetizing is performed by prefixing a message with an `M_PROTO` message. The original message type is stored in the 1 byte data portion of the `M_PROTO` message.

On the read-side, only the `M_PROTO`, `M_PCPROTO`, `M_STOP`, `M_START`, `M_STOPI`, `M_STARTI`, `M_IOCTL`, `M_DATA`, `M_FLUSH`, and `M_READ` messages are packetized. All other message types are passed upstream unmodified.

Since all unread state information is held in the master’s stream head read queue, flushing of this queue is disabled.

On the write-side, all messages are sent down unmodified.

With this module in place, all reads from the master side of the pseudo terminal should be performed with the `getmsg(2)` or `getpmsg()` function. The control part of the message contains the message type. The data part contains the actual data associated with that message type. The onus is on the application to separate the data into its component parts.

SEE ALSO

`getmsg(2)`, `ioctl(2)`, `ldterm(7M)`, `ptem(7M)`, `streamio(7I)`, `termio(7I)`

`STREAMS Programming Guide`
**NAME**  
`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)`

---

386  
man pages section 7: Device and Network Interfaces • Last Revised 20 Mar 1995
pcn – AMD PCnet Ethernet controller device driver

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.

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 therefore supports only the connectionless mode of data transfer. Thus, a DLPI user should issue a DL_ATTACH_REQ primitive to select the device to be used. Valid DLPI primitives are defined in <sys/dlpi.h>. Refer to dlpi(7P) for more information.

The device is initialized on the first attach and de-initialized (stopped) on the last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to a DL_INFO_REQ from the user are 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
pcn(7D)

/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>IA</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/entndnsn  block files
/dev/rdsk/entndnsn  raw files
where:
  cn       controller n
  tn       technology type n
  0x1ROM,0x2OTPROM,0x3EPROM,
  0x4EEPROM,0x5FLASH,0x6SRAM,
  0x7DRAM
  an       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>IA</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 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/pcn dial-in devices
/dev/cua/pcn dial-out devices where: n is the PCMCIA physical
socket number.

SEE ALSO
cu(1C), tip(1), uucp(1C), autopush(1M), pcmciad(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.
The security associations ("SAs") are used to protect both inbound and outbound packets.

A user process (or possibly multiple co-operating processes) maintains SADBs by sending messages over a special kind of socket. This is analogous to the method described in `route(7P)`. Only a superuser may access an SADB.

The operating system may spontaneously emit messages in response to external events, such as a request for a new SA for an outbound datagram, or to report the expiration of an existing SA.

Messages

Messages include:

```c
#define SADB_GETSPI   /* Get a new SPI value from the system. */
#define SADB_UPDATE   /* Update an SA. */
#define SADB_ADD      /* Add a fully-formed SA. */
#define SADB_DELETE   /* Delete an SA. */
#define SADB_GET      /* Get an SA */
#define SADB_ACQUIRE  /* Kernel needs a new SA. */
#define SADB_REGISTER /* Register to receive ACQUIRE messages. */
#define SADB_EXPIRE   /* SA has expired. */
#define SADB_FLUSH    /* Flush all SAs. */
#define SADB_DUMP     /* Get all SAs. (Unreliable) */
#define SADB_X_PROMISC /* Listen promiscuously */
#define SADB_X_PCHANGE /* Passive listener change (passive ACQUIRE) */
```

The base message header consists of:

```c
text struct sadb_msg {
    uint8_t sadb_msg_version; /* Set to PF_KEY_V2, for compatibility */
    uint8_t sadb_msg_type;   /* Message type */
    uint8_t sadb_msg_errno;  /* Why message failed */
    uint8_t sadb_msg_satype; /* Which security service */
    uint16_t sadb_msg_len;   /* Length in 8-byte units */
    uint16_t sadb_msg_reserved; /* Zero out */
    uint32_t sadb_msg_seq;   /* For message originator */
```

---

**pf_key(7P)**

<table>
<thead>
<tr>
<th>NAME</th>
<th>pf_key – security association database</th>
</tr>
</thead>
</table>
| SYNOPSIS    | #include <sys/types.h>  
#include <sys/socket.h>
#include <net/pfkeyv2.h>  
int socket(PF_KEY, SOCK_RAW, PF_KEY_V2); |
| DESCRIPTION | Keying information for IPsec security services is maintained in security association databases “SADBs”. The security associations (“SAs”) are used to protect both inbound and outbound packets.  
A user process (or possibly multiple co-operating processes) maintains SADBs by sending messages over a special kind of socket. This is analogous to the method described in `route(7P)`. Only a superuser may access an SADB.  
The operating system may spontaneously emit messages in response to external events, such as a request for a new SA for an outbound datagram, or to report the expiration of an existing SA.  
One opens the channel for passing SADB control messages by using the socket call shown in the SYNOPSIS section above. More than one key socket can be open per system.  
Messages are formed by a small base header, followed by a number, zero or more, of extension messages, some of which require additional data following them. The base message and all extensions must be eight-byte aligned. An example message is the GET message, which requires the base header, the SA extension, and the ADDRESS_DST extension. |
| Messages    | Messages include:  
```c
#define SADB_GETSPI   /* Get a new SPI value from the system. */
#define SADB_UPDATE   /* Update an SA. */
#define SADB_ADD      /* Add a fully-formed SA. */
#define SADB_DELETE   /* Delete an SA. */
#define SADB_GET      /* Get an SA */
#define SADB_ACQUIRE  /* Kernel needs a new SA. */
#define SADB_REGISTER /* Register to receive ACQUIRE messages. */
#define SADB_EXPIRE   /* SA has expired. */
#define SADB_FLUSH    /* Flush all SAs. */
#define SADB_DUMP     /* Get all SAs. (Unreliable) */
#define SADB_X_PROMISC /* Listen promiscuously */
#define SADB_X_PCHANGE /* Passive listener change (passive ACQUIRE) */
```

The base message header consists of:  
```c
text struct sadb_msg {
    uint8_t sadb_msg_version; /* Set to PF_KEY_V2, for compatibility */
    uint8_t sadb_msg_type;   /* Message type */
    uint8_t sadb_msg_errno;  /* Why message failed */
    uint8_t sadb_msg_satype; /* Which security service */
    uint16_t sadb_msg_len;   /* Length in 8-byte units */
    uint16_t sadb_msg_reserved; /* Zero out */
    uint32_t sadb_msg_seq;   /* For message originator */
``` |
Extension types include:

```c
#define SADB_EXT_SA /* SA information */
#define SADB_EXT_LIFETIME_HARD /* Hard lifetime */
#define SADB_EXT_LIFETIME_SOFT /* Soft lifetime */
#define SADB_EXT_ADDRESS_SRC /* Source address */
#define SADB_EXT_ADDRESS_DST /* Destination address */
#define SADB_EXT_ADDRESS_PROXY /* Proxy address */
#define SADB_EXT_KEY_AUTH /* Authentication key */
#define SADB_EXT_KEY_ENCRYPT /* Encryption key */
#define SADB_EXT_IDENTITY_SRC /* Source certificate ID */
#define SADB_EXT_IDENTITY_DST /* Destination certificate ID */
#define SADB_EXT_SENSITIVITY /* Sensitivity information */
#define SADB_EXT_PROPOSAL /* Security proposal */
#define SADB_EXT_SUPPORTED_AUTH /* Supported authentication algorithms */
#define SADB_EXT_SUPPORTED_ENCRYPT /* Supported encryption algorithms */
#define SADB_EXT_SPIRANGE /* Range of possible SPIs */
```

Extension headers include:

**Generic Extension Header**

```c
struct sadb_ext {
    uint16_t sadb_ext_len; /* In 64-bit words, inclusive */
    uint16_t sadb_ext_type; /* 0 is reserved */
};
```

**Security Association Information Extension**

```c
struct sadb_sa {
    uint16_t sadb_sa_len;
    uint16_t sadb_sa_exttype; /* ASSOCIATION */
    uint32_t sadb_sa_spi;
    uint8_t sadb_sa_replay;
    uint8_t sadb_sa_state;
    uint8_t sadb_sa_auth;
    uint8_t sadb_sa_encrypt;
    uint32_t sadb_sa_flags;
};
```

**Lifetime Extension**

```c
struct sadb_lifetime {
    uint16_t sadb_lifetime_len;
    uint16_t sadb_lifetime_exttype; /* SOFT, HARD, CURRENT */
    uint32_t sadb_lifetime_allocations;
    uint64_t sadb_lifetime_bytes;
    uint64_t sadb_lifetime_addtime;
    uint64_t sadb_lifetime_usetime;
};
```

**Address Extension**
struct sadb_address {
    uint16_t sadb_address_len;
    uint16_t sadb_address_exttype; /* SRC, DST, PROXY */
    uint8_t sadb_address_proto; /* Proto for ports... */
    uint8_t sadb_address_prefixlen; /* Prefix length. */
    uint16_t sadb_address_reserved; /* Padding */
    /* Followed by a sockaddr structure. */
};

Keying Material Extension
struct sadb_key {
    uint16_t sadb_key_len;
    uint16_t sadb_key_exttype; /* AUTH, ENCRYPT */
    uint16_t sadb_key_bits;
    uint16_t sadb_key_reserved;
    /* Followed by actual key(s) in canonical (outbound proc.) order. */
};

Identity Extension
struct sadb_ident {
    uint16_t sadb_ident_len;
    uint16_t sadb_ident_exttype; /* SRC, DST, PROXY */
    uint16_t sadb_ident_type; /* FQDN, USER_FQDN, etc. */
    uint16_t sadb_ident_reserved; /* Padding */
    uint64_t sadb_ident_id; /* For userid, etc. */
    /* Followed by an identity null-terminate C string if present. */
};

Sensitivity/Integrity Extension
struct sadb_sens {
    uint16_t sadb_sens_len;
    uint16_t sadb_sens_exttype; /* SENSITIVITY */
    uint32_t sadb_sens_dpd;
    uint8_t sadb_sens_sens_level;
    uint8_t sadb_sens_sens_len; /* 64-bit words */
    uint8_t sadb_sens_integ_level;
    uint8_t sadb_sens_integ_len; /* 64-bit words */
    uint32_t sadb_sens_reserved;
    /* followed by two uint64_t arrays */
    * uint64_t sadb_sens_bitmap[sens_bitmap_len];
    * uint64_t integ_bitmap[integ_bitmap_len];
    */
};

Proposal Extension
struct sadb_prop {
    uint16_t sadb_prop_len;
    uint16_t sadb_prop_exttype; /* PROPOSAL */
A Combination Instance for a Proposal

```c
struct sadb_comb {
    uint8_t sadb_comb_auth;
    uint8_t sadb_comb_encrypt;
    uint16_t sadb_comb_flags;
    uint16_t sadb_comb_auth_minbits;
    uint16_t sadb_comb_auth_maxbits;
    uint16_t sadb_comb_encrypt_minbits;
    uint16_t sadb_comb_encrypt_maxbits;
    uint32_t sadb_comb_reserved;
    uint32_t sadb_comb_soft_allocations;
    uint32_t sadb_comb_hard_allocations;
    uint64_t sadb_comb_soft_bytes;
    uint64_t sadb_comb_hard_bytes;
    uint64_t sadb_comb_soft_addtime;
    uint64_t sadb_comb_hard_addtime;
    uint64_t sadb_comb_soft_usetime;
    uint64_t sadb_comb_hard_usetime;
};
```

Supported Algorithms Extension

```c
struct sadb_supported {
    uint16_t sadb_supported_len;
    uint16_t sadb_supported_exttype;
    uint32_t sadb_supported_reserved;
};
```

An Algorithm Instance

```c
struct sadb_alg {
    uint8_t sadb_alg_id;    /* Algorithm type. */
    uint8_t sadb_alg_ivlen; /* IV len, in bits */
    uint16_t sadb_alg_minbits; /* Min. key len (in bits) */
    uint16_t sadb_alg_maxbits; /* Max. key length */
    uint16_t sadb_alg_reserved;
};
```

Range of SPIs Extension

```c
struct sadb_spirange {
    uint16_t sadb_spirange_len;
    uint16_t sadb_spirange_exttype;    /* SPI_RANGE */
    uint32_t sadb_spirange_min
    uint32_t sadb_spirange_max;
    uint32_t sadb_spirange_reserved;
};
```
Each message has a behavior. A behavior is defined as where the initial message travels, for example, user to kernel, and what subsequent actions are expected to take place. Contents of messages are illustrated as:

\[ \text{<base, REQUIRED EXTENSION, REQ., (OPTIONAL EXTENSION), (OPT)>} \]

The SA extension is sometimes used only for its SPI field. If all other fields must be ignored, this is represented by SA(*)

The lifetime extensions are represented with one to three letters after the word lifetime, representing (H)ARD, (S)OFT, and (C)URRENT.

The address extensions are represented with one to three letters after the word "address," representing (S)RC, (D)ST, (P)ROXY.

Note that when an error occurs, only the base header is sent. Typical errors include:

- **EINVAL**: Various message improprieties, including SPI ranges that are malformed, weak keys, and others.
- **ENOMEM**: Needed memory was not available.
- **ENGSIZ**: The message exceeds the maximum length allowed.
- **EEXIST**: An SA (that is being added or created with GETSPI) already exists.
- **ESRCH**: An SA could not be found.

The following are examples of message use and behavior:

**SADB_GETSPI**

Send a SADB_GETSPI message from a user process to the kernel.

\[ \text{<base, address, SPI range>} \]

The kernel returns the SADB_GETSPI message to all listening processes.

\[ \text{<base, SA(*), address (SD)>} \]

**SADB_UPDATE**

Send a SADB_UPDATE message from a user process to the kernel.

\[ \text{<base, SA, (lifetime(HS)), address(SD), (address(P), key (AE)), (identity(SD)), (sensitivity)>} \]

The kernel returns the SADB_UPDATE message to all listening processes.

\[ \text{<base, SA(*), address (SD)>} \]

**SADB_ADD**

Send a SADB_ADD message from a user process to the kernel.
The kernel returns the `SADB_ADD` message to all listening processes.

### SADB_DELETE

Send a `SADB_DELETE` message from a user process to the kernel.

The kernel returns the `SADB_DELETE` message to all listening processes.

### SADB_GET

Send a `SADB_GET` message from a user process to the kernel.

The kernel returns the `SADB_GET` message to the socket that sent the `SADB_GET` message.

### SADB_ACQUIRE

The kernel sends a `SADB_ACQUIRE` message to registered sockets. Note that any `GETSPI`, `ADD`, or `UPDATE` calls in reaction to an `ACQUIRE` must fill in the `sadb_msg_seq` of those messages with the one in the `ACQUIRE` message. The address (SD) extensions must have the port fields filled in with the port numbers of the session requiring keys if appropriate. If key management fails, the user process should send an `SADB_ACQUIRE` to indicate failure.

### SADB_REGISTER

Send a `SADB_REGISTER` message from a user process to the kernel.

The kernel returns the `SADB_REGISTER` message to registered sockets, with algorithm types supported by the kernel being indicated in the supported algorithms field. Note that this message may arrive asynchronously due to an algorithm being loaded or unloaded into a dynamically linked kernel.
SADB_EXPIRE
The kernel sends a SADB_EXPIRE message to all listeners when the soft limit of a
security association has been expired.

<base, SA, lifetime (C and one of HS), address (SD)>

SADB_FLUSH
Send a SADB_FLUSH message from a user process to the kernel.

<base> The kernel returns the SADB_FLUSH message to all listening sockets.

<base>

SADB_DUMP
Send a SADB_DUMP message from a user process to the kernel.

<base> Several SADB_DUMP messages will return from the kernel to the sending socket.

<base, SA, lifetime (HSC), address (SD), address (P), key (AE),
(identity (SD),) sensitivity> To mark the end of a dump a single base header
will arrive with its sadb_mdg_seq set to 0.

<base>

SADB_X_PROMISC
Send a SADB_X_PROMISC message from a user process to the kernel.

<base> The kernel returns the SADB_X_PROMISC message to all listening processes.

<base>

SADB_X_PCHANGE
The kernel sends a SADB_X_PCHANGE message to registered sockets. Note that the
address (SD) extensions must have the port fields filled in with the port numbers of
the session requiring keys if appropriate.

<base, address (SD), (identity (SD),)
(sensitivity,)(proposal)>

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>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsrx (64-bit)</td>
</tr>
</tbody>
</table>
### pf_key(7P)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

ipseckey(1M), ipsec(7P), ipsecah(7P), ipsecesp(7P), route(7P)


**NOTES**

Time-based lifetimes may not expire with exact precision in seconds because kernel load may affect the aging of SAs.
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.

pfmod applies the current packet filter to all M_DATA and M_PROTO messages arriving on its read queue. The module prepares these messages for examination by first skipping over all leading M_PROTO message blocks to arrive at the beginning of the message’s data portion. If there is no data portion, pfmod accepts the message and passes it along to its upstream neighbor. Otherwise, the module ensures that the part of the message’s data that the packet filter might examine lies in contiguous memory, calling the pullupmsg(9F) utility routine if necessary to force contiguity. (Note: this action destroys any sharing relationships that the subject message might have had with other messages.) Finally, it applies the packet filter to the message’s data, passing the entire message upstream to the next module if the filter accepts, and discarding the message otherwise. See PACKET FILTERS below for details on how the filter works.

If there is no packet filter yet in effect, the module acts as if the filter exists but does nothing, implying that all incoming messages are accepted. The IOCTLS section below describes how to associate a packet filter with an instance of pfmod.

pfmod passes all other messages through unaltered to its upper neighbor.

pfmod intercepts M_IOCTL messages for the ioctl described below. The module passes all other messages through unaltered to its lower neighbor.

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_FilterLen; /* length of filter cmd list */
    ushort_t Pf_Filter[ENMAXFILTERS]; /* filter command list */
};
```

The Pf_Priority field is included only for compatibility with other packet filter implementations and is otherwise ignored. The packet filter itself is specified in the Pf_Filter array as a sequence of two-byte commands, with the Pf_FilterLen field giving the number of commands in the sequence. This implementation restricts...
the maximum number of commands in a filter (ENMAXFILTERS) to 255. The next
section describes the available commands and their semantics.

**PACKET FILTERS**

A packet filter consists of the filter command list length (in units of ushort_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.
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)
    syserr("pfmod");

/*
 * Set up filter. Offset is the displacement of the Ethernet type field from the beginning of the packet in units of ushort_t.
 */
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];
```

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), le(7D), pullupmsg(9F)
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
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)

### 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.

- **Unable to attach**  
  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 (pm) driver provides an interface for applications to configure devices within the system for Power Management. The interface is provided through `ioctl(2)` commands. The pm driver may be accessed using `/dev/pm`.

The Power Management framework model allows the system to be viewed as a collection of devices. Each device is a collection of components that comprise the smallest power manageable units. The device driver controls the definition of a device’s power manageable components.

A component can either be `busy` or `idle` at the current power level. Normally, the Power Management framework takes an `idle` component to the next lower power level. The Power Management framework uses two factors to determine this transition: the component must have been idle for at least the threshold time, and the device to which the component belongs must satisfy any dependency requirements. A dependency occurs when a device requires another device to be power managed before it can be power managed. Dependencies occur on a per device basis: when a dependency exists, no components of a device may be managed unless all the devices it depends upon are first power managed.

Using the commands below, an application may take control of the Power Management of a device from the Power Management framework driver and manage the transition of device power levels directly.

All of the ioctl commands in this section are obsolete and will be removed in a future release. See the NEW IOCTLS section of this man page for new commands.

For this set of ioctl commands, `arg` (see `ioctl(2)`) points to a structure of type `pm_request` defined in `<sys/pm.h>`:

```c
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; /* holds 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. This may be the name of a device special file or any trailing substring of the physical path to the device.
- **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 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 power managed by framework.
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. This command is obsolete and will be removed in a future release. Use
PM_DIRECT_PM instead.

Error codes:

- **EBUSY**  Device already disabled from being power managed by
framework.
- **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. This command is obsolete and
will be removed in a future release. Use PM_GET_FULL_POWER instead.

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. This command is obsolete and will be removed in a future release. Please
use PM_GET_CURRENT_POWER instead.

Error codes:

- **EINVAL**  Device component out of range.
- **EAGAIN**  Device component level is not currently known.

**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. This
command is obsolete and will be removed in a future release. Use PM_SET_CURRENT_POWER instead.

Error codes:

EINVAL  Device component out of range, or power level < 0.
EIO      Failed to power device or its ancestors or its dependents 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 Power Management by the framework. By default, all configured devices are power managed by the framework. This command is obsolete and will be removed in a future release. Use PM_RELEASE_DIRECT_PM instead.

Error codes:

EINVAL  Device already being power managed by the framework.
EPERM    Caller is neither super-user nor owner of the device.

NEW IOCTLs

The ioctl commands in this section replace the obsolete commands listed above and take a pointer to a different structure and support more complete functionality.

For this set of ioctl commands, arg (see ioctl(2)) points to a structure of type pm_req defined in <sys/pm.h>:

typedef struct pm_req {
  char *physpath;  /* physical path of device to configure */
  int component;  /* the component of the device */
  int value;      /* power level, threshold value, or count */
  void *data;     /* command-dependent variable sized data */
  size_t datasize;  /* size of data buffer */
} pm_req_t;

The fields should contain the following data:

physpath  Pointer to the physical path of a device. See libdevinfo(3). For example, for the device /devices/pseudo/pm@0:pm the physpath value would be /pseudo/pm@0.

component  Non-negative integer specifying which component is being configured. The numbering starts at zero.

value  Non-negative integer specifying the threshold value in seconds or the desired power level, or the number of levels being specified.

data  Pointer to a buffer which contains or receives variable-sized data, such as the name of a device upon which this device has a dependency.
size

Size of the data buffer.

Not all fields are used in each command.

PM_DIRECT_PM

The device named by physpath is disabled from being power managed by the framework. The caller will power manage the device directly using the PM_DIRECT_NOTIFY, PM_GET_TIME_IDLE and PM_GET_CURRENT_POWER, PM_GET_FULL_POWER and PM_SET_CURRENT_POWER commands. If the device needs to have its power level changed either because its driver calls pm_raise_power(9F), pm_lower_power(9F), or pm_power_has_changed(9F) or because the device is the parent of another device that is changing power level or a device that this device depends on is changing power level, then the power level change of the device will be blocked and the caller will be notified as described below for the PM_DIRECT_NOTIFY command.

Error codes:

EBUSY Device already disabled for Power Management by framework.
EPERM Caller is neither superuser nor effective group ID of 0.

PM_RELEASE_DIRECT_PM

The device named by physpath (which must have been the target of a PM_DIRECT_PM command) is re-enabled for Power Management by the framework.

Error codes:

EINVAL Device component out of range.

PM_DIRECT_NOTIFY PM_DIRECT_NOTIFY_WAIT

These commands allow the process that is directly power managing a device to be notified of events that could change the power level of the device. When such an event occurs, this command returns information about the event.

arg (see ioctl(2)) points to a structure of type pm_state_change defined in <sys/pm.h>:

typedef struct pm_state_change {
    char *physpath; /* device which has changed state */
    int component; /* which component changed state */
    #if defined(_BIG_ENDIAN)
        ushort_t flags; /* PSC_EVENT_LOST, PSC_ALL_LOWEST */
        ushort_t event; /* type of event */
    #else
        ushort_t event; /* type of event */
        ushort_t flags; /* PSC_EVENT_LOST, PSC_ALL_LOWEST */
    #endif
    time_t timestamp; /* time of state change */
    int old_level; /* power level changing from */
    int new_level; /* power level changing to */
    size_t size; /* size of buffer physpath points to */
} pm_state_change_t;

When an event occurs, the struct pointed to by arg is filled in.
If the event type is `PSC_PENDING_CHANGE`, then the information in the rest of the struct describes an action that the framework would have taken if the device were not directly power managed by the caller. The caller is responsible for completing the indicated level changes using `PM_SET_CURRENT_POWER` below.

An event type of `PSC_HAS_CHANGED` indicates that the driver for the directly power managed device has called `pm_power_has_changed(9F)` due to the device changing power on its own. It is provided to allow the caller to track the power state of the device.

The system keeps events in a circular buffer. If the buffer overflow, the oldest events are lost and when the event that next follows a lost event is retrieved it will have `PSC_EVENT_LOST` set in flags.

`PM_DIRECT_NOTIFY` returns `EWOULDBLOCK` if no event is pending, and `PM_DIRECT_NOTIFY_WAIT` blocks until an event is available.

`pm` also supports the `poll(2)` interface. When an event is pending a `poll(2)` call that includes a file descriptor for `/dev/pm` and that has `POLLIN` or `POLLRDNORM` set in its event mask will return.

`PM_SET_CURRENT_POWER` Component `component` of the device named by `physpath` (which must contain the physical path of a device against which the process has issued a `PM_DIRECT_PM` command) is set to power level `value`. If all components of the device named by `physpath` were at level 0, `value` is non-zero and some device has a dependency on this device, then all components of that device will be brought to full power before this command returns. Similarly, if the parent of the target device is powered off, then it will be brought up as needed before this command returns. When `PM_SET_CURRENT_POWER` is issued against a device, the resulting power change is included in the event list for `PM_DIRECT_NOTIFY`.

Error codes:

`EINVAL` Device component out of range, or power level < 0.

`EIO` Failed to power device or its ancestors or the devices on which this device has dependency or their ancestors. Note that this may not indicate a failure, the device driver may have rejected the command as inappropriate because the component has become busy.

`EPERM` Caller has not previously issued a successful `PM_DIRECT_PM` command against this device.

`PM_GET_FULL_POWER` The highest supported power level of component `component` of the device named by `physpath` is returned.
PM_GET_CURRENT_POWER
The current power level of component component of the device named by physpath is returned.

Error codes:
EAGAIN Device component power level is not currently known.

PM_GET_TIME_IDLE
PM_GET_TIME_IDLE returns the number of seconds that component component of the device named by physpath has been idle. If the device is not idle, then 0 is returned.

Note that because the state of the device may change between the time the process issues the PM_GET_TIME_IDLE command and the time the process issues a PM_SET_CURRENT_POWER command to reduce the power level of an idle component, the process must be prepared to deal with a PM_SET_CURRENT_POWER command returning failure because the driver has rejected the command as inappropriate because the device component has become busy. This can be differentiated from other types of failures by issuing the PM_GET_TIME_IDLE command again to see if the component has become busy.

ERRORS
Upon error, the commands will return −1, and set errno. In addition to the error codes listed above by command, the following error codes are common to all commands:

EFAULT Bad address passed in as argument.
ENODEV Device is not power manageable, or device is not configured.
ENXIO Too many opens attempted.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface stability</td>
<td>Unstable (Interfaces under OBSOLETE IOCTLs are obsolete.)</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_idle_component(9F), pm_lower_power(9F), pm_power_has_changed(9F), pm_raise_power(9F),

Writing Device Drivers
NAME | poll – driver for fast poll on many file descriptors
SYNOPSIS | #include <sys/devpoll.h>
PARAMETERS | 
| fd | Open file descriptor that refers to the /dev/poll driver.
| path | /dev/poll
| buf | Array of pollfd structures.
| bufsize | Size of buf in bytes.
| arg | Pointer to pollcall structure.
| pfd | Pointer to pollfd structure.
DESCRIPTION | The /dev/poll driver is a special driver that lets users monitor multiple sets of polled file descriptors. By using the /dev/poll driver, users can poll large numbers of file descriptors very efficiently. Access to the /dev/poll driver is provided through open(2), write(2), and ioctl(2) system calls.

Writing an array of pollfd struct to the /dev/poll driver has the effect of adding these file descriptors to the monitored poll file descriptor set represented by the fd. Users wishing to monitor multiple file descriptor sets should open the /dev/poll driver multiple times. Each fd corresponds to one set. For each pollfd struct entry (defined in sys/poll.h):

```c
struct pollfd {
    int fd;
    short events;
    short revents;
}
```

The fd field specifies the file descriptor being polled. The events field indicates the interested poll events on the file descriptor. If a pollfd array contains multiple pollfd entries with same fd field, the "events" field in each pollfd entry is OR'ed. A special POLLREMOVE event in the events field of the pollfd structure will remove the fd from the monitored set. The revents field is not used. Write returns the number of bytes written successfully or -1 when write fails.

The DP_POLL ioctl is used to retrieve returned poll events occurred on the polled file descriptors in the monitored set represented by fd. arg is a pointer to the devpoll structures which are defined as follows:

```c
struct dvpoll {
    struct pollfd* dp_fds;
    int dp_nfds;
    int dp_timeout;
}
```

The dp_fds points to a buffer which is used to hold an array of returned pollfd structures. The dp_nfds field specifies the size of the buffer in terms of the number of pollfd entries it contains; dp_nfds also indicates the maximum number of file descriptors on which a user is interested in getting poll information. If there is no interested events on any of the polled file descriptors, the DP_POLL ioctl call will wait dp_timeout milliseconds before returning. If dp_timeout is 0, the ioctl call
returns immediately; if \( dp_{\text{timeout}} \) is -1, the call blocks until an interested poll events is available or the call is interrupted. Upon return, if the ioctl call has failed, -1 is returned. The memory content pointed by \( dp_{\text{fds}} \) is not modified. A return value 0 means the ioctl is timed out. In this case, the memory content pointed by \( dp_{\text{fds}} \) is not modified. If the call is successful, it returns the number of valid pollfd entries in the array pointed by \( dp_{\text{fds}} \); the contents of the rest of the buffer is undefined. For each valid pollfd entry, the \( fd \) field indicates the file descriptor on which the polled events happened. The \( \text{events} \) field is the user specified poll events. The \( \text{revents} \) field contains the events occurred. -1 is returned if the call fails.

\( \text{DP\_ISPOLLED} \) ioctl allows user to query if a file descriptor is already in the monitored set represented by \( fd \). The \( fd \) field of the pollfd structure indicates the file descriptor of interest. The \( \text{DP\_ISPOLLED} \) ioctl returns 1 if the file descriptor is in the set. The \( \text{events} \) field contains the currently polled events. The \( \text{revents} \) field contains 0. The ioctl returns 0 if the file descriptor is not in the set. The pollfd structure pointed by \( pfld \) is not modified. The ioctl returns a -1 if the call fails.

EXAMPLES

EXAMPLE 1 The following example shows how /dev/poll may be used.

```c
{ ... */
  /* open the driver */
  /*
    if ((wfd = open("/dev/poll", O_RDWR)) < 0) {
      exit(-1);
    }
    pollfd = (struct pollfd* )malloc(sizeof(struct pollfd) * MAXBUF);
    if (pollfd == NULL) {
      close(wfd);
      exit(-1);
    }
  */
  /* initialize buffer */
  /*
    for (i = 0; i < MAXBUF; i++) {
      pollfd[i].fd = fds[i];
      pollfd[i].events = POLLIN;
      pollfd[i].revents = 0;
    }
    if (write(wfd, &pollfd[0], sizeof(struct pollfd) * MAXBUF) !=
        sizeof(struct pollfd) * MAXBUF) {
      perror("failed to write all pollfds");
      close (wfd);
      free(pollfd);
      exit(-1);
    }
  */
  /* read from the devpoll driver */
  dopoll.dp_timeout = -1;
  dopoll.dp_nfds = MAXBUF;
  dopoll.dp_fds = pollfd;
}```
EXAMPLE 1 The following example shows how /dev/poll may be used.

```c
result = ioctl(wfd, DP_POLL, &dopoll);
if (result < 0) {
    perror("/dev/poll ioctl DP_POLL failed");
    close(wfd);
    free(pollfd);
    exit(-1);
}
for (i = 0; i < result; i++) {
    read(dopoll.dp_fds[i].fd, rbuf, STRLEN);
}
```

... 

ERRORS

- **EACCES** A process does not have permission to access the content cached in /dev/poll.
- **EINVAL** The request or arg parameter is not valid for this device.
- **ENXIO** The O_NONBLOCK flag is set, the named file is a FIFO, the O_WRONLY flag is set, and no process has the file open for reading; or the named file is a character special or block special file and the device associated with this special file does not exist.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC, Intel</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcarx.u, SUNWcsxu (64-bit Solaris)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsr, SUNWcsu (32-bit Solaris on Intel)</td>
</tr>
<tr>
<td></td>
<td>SUNWhea (header files)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

open(2), poll(2), write(2), attributes(5)

NOTES

The /dev/poll API is particularly beneficial to applications which poll a large number of file descriptors and poll them repeatedly. Applications will exhibit the best performance gain if the polled file descriptor list rarely change.
When using the `/dev/poll` driver, user should pay attention to remove a closed file descriptor from a monitored poll set. Failure to do so may result in a `POLLNVAL` `revents` being returned for the closed file descriptor. When a file descriptor is closed but not removed from the monitored set, and if the file descriptor is reused in subsequent open of a possibly different device, user will be polling the device associated with the reused file descriptor. In a multithreaded application, careful coordination among threads doing close and `DP_POLL ioctl` is recommended for consistent results.

The `/dev/poll` driver caches a list of polled file descriptors, which are specific to a process. Therefore, the `/dev/poll` file descriptor of a process will be inherited by its child process, just like any other file descriptors. But the child process will have very limited access through this inherited `/dev/poll` file descriptor. Any attempt to write or do ioctl by the child process will result in an `EACCES` error. The child process should close the inherited `/dev/poll` file descriptor and open its own if desired.

The `/dev/poll` driver does not yet support polling. Polling on a `/dev/poll` file descriptor will result in `POLLERR` being returned in the `revents` field of `pollfd` structure.
ppp(7M)

**NAME**
ppp, ppp_diag, ipd, ipdptp, ipdcm – STREAMS modules and drivers for the Point-to-Point Protocol

**DESCRIPTION**

Note – ppp has been superseded and may be removed from future versions of Solaris. Refer to `pppd(1M)` instead. See `/etc/ppp/` for configuration files. Use `man pppd` for more information.

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 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.

**Operation**

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-ipoint
          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), pppd(1M), attributes(5)
ptem(7M)

**NAME**  
ptem – STREAMS Pseudo Terminal Emulation module

**SYNOPSIS**  
```c
int ioctl(int 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 TCSSETA, TCSSETAF, TCSSETAW, TGETA, TCSETS, TCSETSW, TCSETSF, TGETS, TCSBRK, JWINSIZE, TIOCGWINSZ, and TIOCWSWINSZ 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**  
```c
<sys/ptem.h>
```

**SEE ALSO**  
stty(1), ioctl(2), ldterm(7M), pckt(7M), streamio(7I), termio(7I)

STREAMS Programming Guide
ptm – STREAMS pseudo-tty master driver

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 getmmsg() 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

Device and Network Interfaces 419
STREAMS pseudo-tty slave driver

The pseudo-tty subsystem simulates a terminal connection, where the master side represents the terminal and the slave represents the user process’s special device end point. In order to use the pseudo-tty subsystem, a node for the master side driver /dev/ptmx and N nodes for the slave driver (N is determined at installation time) must be installed. The names of the slave devices are /dev/pts/M where M has the values 0 through N-1. When the master device is opened, the corresponding slave device is automatically locked out. No user may open that slave device until its permissions are adjusted and the device unlocked by calling functions grantpt(3C) and unlockpt(3C). The user can then invoke the open system call with the name that is returned by the ptsname(3C) function. See the example below.

Only one open is allowed on a master device. Multiple opens are allowed on the slave device. After both the master and slave have been opened, the user has two file descriptors which are end points of a full duplex connection composed of two streams automatically connected at the master and slave drivers. The user may then push modules onto either side of the stream pair. The user needs to push the ptem(7M) and ldtterm(7M) modules onto the slave side of the pseudo-terminal subsystem to get terminal semantics.

The master and slave drivers pass all messages to their adjacent queues. Only the M_FLUSH needs some processing. Because the read queue of one side is connected to the write queue of the other, the FLUSHR flag is changed to the FLUSHW flag and vice versa. When the master device is closed an M_HANGUP message is sent to the slave device which will render the device unusable. The process on the slave side gets the errno EIO when attempting to write on that stream but it will be able to read any data remaining on the stream head read queue. When all the data has been read, read returns 0 indicating that the stream can no longer be used. On the last close of the slave device, a 0-length message is sent to the master device. When the application on the master side issues a read() or getmsg() and 0 is returned, the user of the master device decides whether to issue a close() that dismantles the pseudo-terminal subsystem. If the master device is not closed, the pseudo-tty subsystem will be available to another user to open the slave device. Since 0-length messages are used to indicate that the process on the slave side has closed and should be interpreted that way by the process on the master side, applications on the slave side should not write 0-length messages. If that occurs, the write returns 0, and the 0-length message is discarded by the ptem module.

The standard STREAMS system calls can access the pseudo-tty devices. The slave devices support the O_NDELAY and O_NONBLOCK flags.

### EXAMPLES

**EXAMPLE 1**

```c
int fdm fds;
char *slavename;
extern char *ptsname();

fdm = open="/dev/ptm", O_RDWR); /* open master */
```
EXAMPLE 1  (Continued)

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
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 termios(7) 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 termios(7) 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 termios(7I) interface.

Instead of having a hardware interface and associated hardware that supports the terminal functions, the functions are implemented by another process manipulating the controller device of the pseudo-terminal.

The controller and the slave devices of the pseudo-terminal are tightly connected. Any data written on the controller device is given to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the controller device (rather than being transmitted from a UAR).

By default, 48 pseudo-terminal pairs are configured as follows:

```
/dev/pty[p-r][0-9a-f] controller devices
/dev/tty[p-r][0-9a-f] slave devices
```

The standard set of termio ioctl's 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 ioctl's 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 `VSTART` member of the `c_cc` array is `^Q`.
  - `TIOCPKT_NOSTOP` whenever XON/XOFF flow control is disabled after being enabled.
- **TIOCREMOTE**: The argument is a pointer to an `int`. If the value of the `int` is non-zero, `remote` mode is enabled; if the value of the `int` is zero, remote mode is disabled. This mode can be enabled or disabled independently of packet mode. When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and not input edited (regardless of the mode the slave side of the pseudo-terminal). Each write to the controller...
device produces a record boundary for the process reading the slave device. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an \textit{EOF} character. Note: this means that a process writing to a pseudo-terminal controller in \textit{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 \texttt{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 \texttt{NEWLINE} characters had been typed (as if, for example, a user had typed the \texttt{LNEXT} character before typing all but the last of those \texttt{NEWLINE} characters). Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

\textbf{EXAMPLES}

\texttt{EXAMPLE 1}

\begin{verbatim}
#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 */
\end{verbatim}

\textbf{FILES}

\begin{verbatim}
/dev/pty[p-z][0-9a-f]    pseudo-terminal controller devices
/dev/tty[p-z][0-9a-f]    pseudo-terminal slave devices
\end{verbatim}

\textbf{SEE ALSO}

\texttt{rlogin(1), rlogind(1M), ldterm(7M), termio(7I), ttcompat(7M),}

\textbf{NOTES}

It is apparently not possible to send an \texttt{EOT} by writing zero bytes in \texttt{TIOCREMOTE} mode.
qe(7D)

NAME
qe – QEC/MACE Ethernet device driver

SYNOPSIS
#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.
- 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).

Device and Network Interfaces  425
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_BOUND 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.

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 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 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), le(7D), qec(7D)
### qec(7D)

<table>
<thead>
<tr>
<th>NAME</th>
<th>qec – QEC bus nexus device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>The qec 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.</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>qe(7D)</td>
</tr>
</tbody>
</table>
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.

The SUNW,qfe device provides a 100Base-TX networking interface. There are two types of SUNW,qfe device: one supporting Sbus and the other supporting the PCI bus interface. The Sbus SUNW,qfe device uses Sun's FEPS ASIC, which provides the Sbus interface and MAC functions. The PCI SUNW,qfe device uses Sun's PFEX ASIC to provide the PCI interface and MAC functions. Both connect with the 100Base-TX on-board transceiver, which connects to a RJ45 connector to provide the Physical layer functions and external connection.

The 100Base-TX standard specifies an “auto-negotiation” protocol to automatically select the mode and speed of operation. The internal transceiver is capable of doing auto-negotiation with the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the Highest Common Denominator mode of operation based on the priorities. It also supports forced-mode of operation where the driver can select the mode of operation.

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

The qfe driver is a “style 2” data link service provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in <sys/dlpi.h>. Refer to dlpi(7P) for more information. An explicit DL_ATTACH_REQ message by the user is required to associate the opened stream with a particular device (ppa). The ppa ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the ppa field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are as follows:

- The maximum SDU is 1500 (ETHERMTU - defined in <sys/ethernet.h>).
- The minimum SDU is 0.
- The dlsap address length is 8.
- The MAC type is DL_ETHER.
The sap length values is -2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.

The service mode is DL_CLDLS.

No optional quality of service (QOS) support is included at present so the QOS fields are 0.

The provider style is DL_STYLE2.

The version is DL_VERSION_2.

The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF).

Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular service access pointer SAP with the stream. The qfe driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type" therefore valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If the user selects a sap with a value of 0, the receiver will be in "802.3 mode". All frames received from the media having a "type" field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open streams which are bound to sap value 0. If more than one stream is in "802.3 mode" then the frame will be duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ if the sap value is 0, and if the destination type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The qfe driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should not hardcode to this particular implementation-specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user may transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the qfe driver. The qfe driver will route received Ethernet frames up all those open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.
In addition to the mandatory connectionless DLPI message set the driver also supports the following primitives.

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable or disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The driver accepts these primitives in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS flag set in the dl_level field enables or disables reception of all frames on the media (“promiscuous mode”), including frames generated by the local host.

When used with the DL_PROMISC_SAP flag set this enables or disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set this enables or disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be root. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive in that it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

By default, the qfe driver performs “auto-negotiation” to select the mode and speed of the link.

The link can be in one of the four following modes:

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

These speeds and modes are described in the 100Base-TX standard.

The auto-negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Speed (100 Mbps or 10 Mbps)
The auto-negotiation protocol does the following:
- Gets all the modes of operation supported by the Link Partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities.
- The highest priority is given to the 100 Mbps, full-duplex; lowest priority is given to 10 Mbps, half-duplex.

The 100Base-TX transceiver is capable of all of the operating speeds and modes listed above. By default, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the link partner.

Sometimes, the user may want to select the speed and mode of the link. The SUNW,qfe device supports programmable "IPG" (Inter-Packet Gap) parameters ipg1 and ipg2. By default, the driver sets ipg1 to 8 byte-times and ipg2 to 4 byte-times (which are the standard values). Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 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
- link partner capabilities

The local transceiver has two sets of capabilities: one set reflects the capabilities of the hardware, which are read-only (RO) parameters, and the second set, which reflects the values chosen by the user, is used in speed selection. There are read/write (RW) capabilities. At boot time, these two sets of capabilities will be the same. The Link Partner capabilities are also read-only parameters because the current default value of these parameters can only be read and cannot be modified.

```
FILES
/dev/qfe  qfe special character device
/kernel/drv/qfe.conf  system wide default device driver properties

SEE ALSO
ndd(1M), netstat(1M), driver.conf(4), dlpi(7P), le(7D)
```
qlc – ISP2200 Family Fibre Channel host bus adapter driver

SYNOPSIS
SUNW, qlc@pci-slot

DESCRIPTION
The qlc host bus adapter driver is a Sun Fibre Channel transport layer-compliant nexus driver for the Qlogic ISP2200/ISP2200A/ISP 2202 adapters. These adapters support Fibre Channel SCSI and IP Protocols, FC-AL public loop profile, point-to-point fabric connection and Fibre Channel service classes two and three.

The qlc driver interfaces with the Sun Fibre Channel transport layer to support the standard functions provided by the SCSA interface. It supports auto request sense and tagged queueing by default. The driver requires that all devices have unique hard addresses in private loop configurations. Devices with conflicting hard addresses will not be accessible.

FILES
/kernel/drv/qlc
   ELF kernel module
/kernel/drv/sparcv9/qlc
   64-bit ELF kernel module
/kernel/drv/qlc.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>
<tr>
<td>Availability</td>
<td>SUNWqlc</td>
</tr>
</tbody>
</table>

SEE ALSO
prtconf(1M), driver.conf(4), fcp(7D), fp(7D)

Writing Device Drivers

ANSI X3.230:1994, Fibre Channel Physical Signaling (FC-PH)

Project 1134-D, Fibre Channel Generic Services (FC-GS-2)

ANSI X3.269-1996, Fibre Channel Arbitrated Loop (FC-AL)

ANSI X3.270-1996, Fibre Channel Protocol for SCSI (FCP-SCSI)

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM)

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Fabric Loop Attachment (FC-FLA)

ISP2200 Firmware Interface Specification, QLogic Corporation
quotactl(7I)

NAME

quotactl – manipulate disk quotas

SYNOPSIS

#include <sys/fs/ufs_quota.h>

int ioctl(int fd, 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.

Q_SETQLIM

Set disk quota limits for user uid. addr is a pointer to a dqblk
structure (defined in sys/fs/ufs_quota.h). This call is
restricted to the super-user.

Q_SYNC

Update the on-disk copy of quota usages for this file system. addr
and uid are ignored.

Q_ALLSYNC

Update the on-disk copy of quota usages for all file systems with
active quotas. addr and uid are ignored.

RETURN VALUES

This ioctl() returns:

0 on success.

-1 on failure and sets errno to indicate the error.
ERRORS

EFAULT  addr is invalid.
EINVAL   The kernel has not been compiled with the QUOTA option. op is invalid.
ENOENT   The quotas file specified by addr does not exist.
EPERM    The call is privileged and the caller was not the super-user.
ESRCH    No disk quota is found for the indicated user. Quotas have not been turned on for this file system.
EUSERS   The quota table is full.

If op is Q_QUOTAON, ioctl() may set errno to:
EACCES   The quota file pointed to by addr exists but is not a regular file. The quota file pointed to by addr exists but is not on the file system pointed to by special.
EIO       Internal I/O error while attempting to read the quotas file pointed to by addr.

FILES
/usr/include/sys/fs/ufs_quota.h
quota-related structure/function definitions and defines

SEE ALSO
quota(1M), quotacheck(1M), quotaon(1M), getrlimit(2), mount(2)

BUGS
There should be some way to integrate this call with the resource limit interface provided by setrlimit() and getrlimit(2).
This call is incompatible with Melbourne quotas.
rns_smt(7D)

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_snt 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
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 superuser.

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, that is, the packet is forwarded.

When routing a packet, the kernel attempts to find the most specific route matching the destination. If no entry is found, the destination is declared to be unreachable, and a routing-miss message is generated if there are any listeners on the routing control socket (described below). If there are two different mask and value-under-the-mask pairs that match, the more specific is the one with more bits in the mask. A route to a host is regarded as being supplied with a mask of as many ones as there are bits in the destination.

A wildcard routing entry is specified with a zero destination address value, and a mask of all zeroes. Wildcard routes are used when the system fails to find other routes matching the destination. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.
One opens the channel for passing routing control messages by using the socket call 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 `sockaddr`, whose length depend on the address family. `sockaddr` 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 `protocol` parameter specifies which messages an application listening on the routing socket is interested in seeing, based on the the address family of the `sockaddr` present. Currently, you can specify `AF_INET` and `AF_INET6` to filter the messages seen by the listener, or alternatively, you can specify `AF_UNSPEC` to indicate that the listener is interested in all routing messages.

The kernel 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
- `ENOBUFS` 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(3SOCKET)` 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(3SOCKET)` 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.

User processes can obtain information about the routing entry to a specific destination by using a `RTM_GET` message.

Messages include:
A message header consists of:

```c
struct rt_msghdr {
    ushort_t rtm_msglen; /* to skip over non-understood messages */
    uchar_t rtm_version; /* future binary compatibility */
    uchar_t rtm_type; /* message type */
    ushort_t rtm_index; /* index for associated ifp */
    pid_t rtm_pid; /* identify sender */
    int rtm_addrs; /* bitmask identifying sockaddrs in msg */
    int rtm_seq; /* for sender to identify action */
    int rtm_errno; /* why failed */
    int rtm_flags; /* flags, incl kern & message, e.g., DONE */
    uint_t rtm_use; /* from rtentry */
    uint_t rtm_inits; /* which values we are initializing */
    struct rt_metrics rtm_rmx; /* metrics themselves */
};
```

where

```c
struct rt_metrics {
    uint32_t rmx_locks; /* Kernel must leave these values alone */
    uint32_t rmx_mtu; /* MTU for this path */
    uint32_t rmx_hopcount; /* max hops expected */
    uint32_t rmx_expire; /* lifetime for route, e.g., redirect */
    uint32_t rmx_recvpipe; /* inbound delay-bandwidth product */
    uint32_t rmx_sendpipe; /* outbound delay-bandwidth product */
    uint32_t rmx_ssthresh; /* outbound gateway buffer limit */
    uint32_t rmx_rtt; /* estimated round trip time */
    uint32_t rmx_rttvar; /* estimated rtt variance */
    uint32_t rmx_pkssent; /* packets sent using this route */
};
```

/* Flags include the values */

```c
#define RTF_UP 0x1 /* route usable */
#define RTF_GATEWAY 0x2 /* destination is a gateway */
#define RTF_HOST 0x4 /* host entry (net otherwise) */
#define RTF_REJECT 0x8 /* host or net unreachable */
#define RTF_DYNAMIC 0x10 /* created dynamically (by redirect) */
#define RTF_MODIFIED 0x20 /* modified dynamically (by redirect) */
#define RTF_DONE 0x40 /* message confirmed */
```
route(7P)

#define RTF_MASK 0x80 /* subnet mask present */
#define RTF_CLONING 0x100 /* generate new routes on use */
#define RTF_XRESOLVE 0x200 /* external daemon resolves name */
#define RTF_LLINFO 0x400 /* generated by ARP */
#define RTF_STATIC 0x800 /* manually added */
#define RTF_BLACKHOLE 0x1000 /* just discard pkts (during updates) */
#define RTF_PRIVATE 0x2000 /* do not advertise this route */
#define RTF_PROTO2 0x4000 /* protocol specific routing flag #2 */
#define RTF_PROTO1 0x8000 /* protocol specific routing flag #1 */

/* Specifiers for metric values in rmx_locks and rtm_inits are */
#define RTV_MTU 0x1 /* init or lock _mtu */
#define RTV_HOPCOUNT 0x2 /* init or lock _hopcount */
#define RTV_EXPIRE 0x4 /* init or lock _expire */
#define RTV_RPIPE 0x8 /* init or lock _recvpipe */
#define RTV_SPIPE 0x10 /* init or lock _sendpipe */
#define RTV_SSTHRESH 0x20 /* init or lock _ssthresh */
#define RTV_RTT 0x40 /* init or lock _rtt */
#define RTV_RTTVAR 0x80 /* init or lock _rttvar */

/* Specifiers for which addresses are present in the messages are */
#define RTA_DST 0x1 /* destination sockaddr present */
#define RTA_GATEWAY 0x2 /* gateway sockaddr present */
#define RTA_NETMASK 0x4 /* netmask sockaddr present */
#define RTA_GENMASK 0x8 /* cloning mask sockaddr present */
#define RTA_IFP 0x10 /* interface name sockaddr present */
#define RTA_IFA 0x20 /* interface addr sockaddr present */
#define RTA_AUTHOR 0x40 /* sockaddr for author of redirect */
#define RTA_BRD 0x80 /* for NEWADDR, broadcast or p-p dest addr */

SEE ALSO ioctl(2), setsockopt(3SOCKET), shutdown(3SOCKET), routing(7P)

NOTES Some of the metrics may not be implemented and return zero. The implemented metrics are set in rtm_inits.
The network facilities provide general packet routing. The routing interface described here can be used to maintain the system's IPv4 routing table. It has been maintained for compatibility with older applications. The recommended interface for maintaining the system's routing tables is the routing socket, described at route(7P). The routing socket can be used to manipulate both the IPv4 and IPv6 routing tables of the system. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two ioctl(2) commands, SIOCADDRT and SIOCDELRRT. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in /usr/include/net/route.h:

```
struct 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 */
/*
 * The kernel does not use this field, and without it the structure is
 * datamodel independent.
 */
#if !defined(_KERNEL)
    struct ifnet *rt_ifp;    /* the answer: interface to use */
#endif /* !defined(_KERNEL) */
};
```

with rt_flags defined from:

```
#define RTF_UP 0x1    /* route usable */
#define RTF_GATEWAY 0x2    /* destination is a gateway */
#define RTF_HOST 0x4    /* host entry (net otherwise) */
```

There are three types of routing table entries: those for a specific host, those for all hosts on a specific network, and those for any destination not matched by entries of the first two types, called a wildcard route. Each network interface installs a routing table entry when it is initialized. Normally the interface specifies if the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same
host specified in the packet. Otherwise, the interface may be requested to address the
packet to an entity different from the eventual recipient; essentially, the packet is
forwarded.

Routing table entries installed by a user process may not specify the hash, reference
count, use, or interface fields; these are filled in by the routing routines. If a route is in
use when it is deleted, meaning its rt_refcnt is non-zero, the resources associated
with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the /dev/ip device.

The rt_use field contains the number of packets sent along the route. This value is used
to select among multiple routes to the same destination. When multiple routes to the
same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard
routes are used only when the system fails to find a route to the destination host and
network. The combination of wildcard routes and routing redirects can provide an
economical mechanism for routing traffic.

<table>
<thead>
<tr>
<th>ERRORS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEXIST</td>
<td>A request was made to duplicate an existing entry.</td>
</tr>
<tr>
<td>ESRCH</td>
<td>A request was made to delete a non-existent entry.</td>
</tr>
<tr>
<td>ENOBUFFS</td>
<td>Insufficient resources were available to install a new route.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Insufficient resources were available to install a new route.</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>The gateway is not directly reachable. For example, it does not match the destination/subnet on any of the network interfaces.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/ip</td>
<td>IP device driver</td>
</tr>
</tbody>
</table>

SEE ALSO route(1M), ioctl(2), route(7P)
NAME | sad – STREAMS Administrative Driver

SYNOPSIS
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/sad.h>
#include <sys/stropts.h>

int ioctl(int fi ld es, int command, int arg);

DESCRIPTION
The STREAMS Administrative Driver provides an interface for applications to perform administrative operations on STREAMS modules and drivers. The interface is provided through ioctl(2) commands. Privileged operations may access the sad driver using /dev/sad/admin. Unprivileged operations may access the sad driver using /dev/sad/user.

The fi ld es argument is an open file descriptor that refers to the sad driver. The command argument determines the control function to be performed as described below. The arg argument represents additional information that is needed by this command. The type of arg depends upon the command, but it is generally an integer or a pointer to a command-specific data structure.

COMMAND FUNCTIONS
The autopush facility (see autopush(1M)) allows one to configure a list of modules to be automatically pushed on a stream when a driver is first opened. Autopush is controlled by the following commands:

SAD_SAP  Allows the administrator to configure the given device’s autopush information. arg points to a strapush structure, which contains the following members:

unit_t  ap_cmd;
major_t  sap_major;
minor_t  sap_minor;
minor_t  sap_lastminor;
unit_t  sap_npush;
unit_t  sap_list [MAXAPUSH] [FMNAMESZ + 1];

The sap_cmd field indicates the type of configuration being done. It may take on one of the following values:

SAP_ONE  Configure one minor device of a driver.
SAP_RANGE  Configure a range of minor devices of a driver.
SAP_ALL  Configure all minor devices of a driver.
SAP_CLEAR  Undo configuration information for a driver.

The sap_major field is the major device number of the device to be configured. The sap_minor field is the minor device number of the device to be configured. The sap_lastminor field is used...
only with the SAP_RANGE command, which configures a range of minor devices between sap_minor and sap_lastminor, inclusive. The minor fields have no meaning for the SAP_ALL command. The sap_npush field indicates the number of modules to be automatically pushed when the device is opened. It must be less than or equal to MAXAPUSH, defined in sad.h. It must also be less than or equal to NSTRPUSH, the maximum number of modules that can be pushed on a stream, defined in the kernel master file. The field sap_list is an array of NULL-terminated module names to be pushed in the order in which they appear in the list.

When using the SAP_CLEAR command, the user sets only sap_major and sap_minor. This will undo the configuration information for any of the other commands. If a previous entry was configured as SAP_ALL, sap_minor should be set to zero. If a previous entry was configured as SAP_RANGE, sap_minor should be set to the lowest minor device number in the range configured.

On failure, errno is set to the following value:

- EFAULT: arg points outside the allocated address space.
- EINVAL: The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.
- ENOENT: The major device number does not represent a STREAMS driver.
- EEXIST: The major-minor device pair is already configured.
- ERANGE: The command is SAP_RANGE and sap_lastminor is not greater than sap_minor, or the command is SAP_CLEAR and sap_minor is not equal to the first minor in the range.
- ENODEV: The command is SAP_CLEAR and the device is not configured for autopush.
- ENOSR: An internal autopush data structure cannot be allocated.

SAD_GAP: Allows any user to query the sad driver to get the autopush configuration information for a given device. arg points to a strapush structure as described in the previous command.
The user should set the sap_major and sap_minor fields of the 
strapush structure to the major and minor device numbers, 
respectively, of the device in question. On return, the strapush 
structure will be filled in with the entire information used to 
configure the device. Unused entries in the module list will be 
zero-filled.

On failure, errno is set to one of the following values:
EFAULT    arg points outside the allocated address space.
EINVAL    The major device number is invalid.
ENOSTR    The major device number does not represent a 
STREAMS driver.
ENODEV    The device is not configured for autopush.

SAD_VML    Allows any user to validate a list of modules (that is, to see if they 
are installed on the system). arg is a pointer to a str_list 
structure with the following members:

int     sl_nmods;
struct str_mlist  *sl_modlist;    The str_mlist structure has 
the following member:

char   l_name[FMNAMESZ+1];

sl_nmods indicates the number of entries the user has allocated 
in the array and sl_modlist points to the array of module 
names. The return value is 0 if the list is valid, 1 if the list contains 
an invalid module name, or −1 on failure. On failure, errno is set 
to one of the following values:
EFAULT    arg points outside the allocated address space.
EINVAL    The sl_nmods field of the str_list 
structure is less than or equal to zero.

SEE ALSO intro(2), ioctl(2), open(2)
STREAMS Programming Guide

DIAGNOSTICS Unless otherwise specified, the return value from ioctl() is 0 upon success and −1 
upon failure with errno set as indicated.
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 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 Solaris IA Device Configuration Guide or the Solaris IA 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.

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.

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_U Law. 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
/usr/demo/SOUND audio demonstration programs

**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>IA</td>
</tr>
</tbody>
</table>

**SEE ALSO**

audioconvert(1), ioctl(2), attributes(5), audio(7I), streamio(7I)
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.
scman – SUNW,scman Sun Fire 15000 management network device driver

The scman (7D) network device driver is a loadable, clonable, STREAMS hardware driver that supports the dlpi connectionless data link provider interface over the SUNW,scman network controller. The scman controller provides a highly available, secure communication channel between the Sun Fire 15000 system controller and dynamic system domains.

The scman driver provides basic support for the SUNW,scman controller. Driver functions include network initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The scman controller is physically located in the Sun Fire 15000 system controller and connects to SUNWdman controllers found in each active dynamic system domain in a Sun Fire 15000 chassis. See dman(7D). All links are point-to-point and are internal to the Sun Fire 15000 chassis. Traffic between the system controller and the dynamic system domains is not accessible by any third party; for example, another system domain within the Sun Fire 15000 chassis. All system domains are accessible through the scman controller.

The link layer frame format is identical to that used by Ethernet (sys/ethernet.h).

The /dev/scman cloning character-special device is used to access the SUNW,scman controller installed on the system.

The scman driver is a style 2 data link provider interface. 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 or physical point of attachment (PPA).

The PPA ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. The only valid unit number is 0. 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 the first attach and deinitialized (stopped) upon the last detach.

The values returned by the scman driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are:

- Maximum service data units (SDU) are 1500.
- Minimum SDU is 0.
- Data link service access point (DLSAP) address length is 8.
- Media access control (MAC) type is DL_ETHER.
- Service access point (SAP) length value is -2, meaning the physical address component is followed immediately by a two-byte SAP component within the
Once in the DL_ATTACHED state, you can transmit DL_BIND_REQ to associate a particular SAP with the stream. The scman driver interprets the SAP field within the DL_BIND_REQ as an Ethernet type; as a result, valid values for the SAP field are in the 0 through 0xFFFF range. Only one Ethernet type can be bound to the stream at any time.

If you select the 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 from 0 through 1500 are assumed to be 802.3 frames and are routed up all open streams that are bound to SAP value 0. If more than one stream is in 802.3 mode, then the frame is duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the DL_BIND_REQ SAP field to determine if the SAP value is 0 and the destination type field is in the range from 0 through 1500. If either is true, the driver computes the length of the message, not including the initial M_PROTO message block (mblk), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The scman driver DLSAP address format consists of the six-byte physical (Ethernet) address component followed immediately by the two-byte SAP (type) component producing an eight-byte DLSAP address. Applications should not be hard-coded to this implementation-specific DLSAP address format, but 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, you can transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the scman driver. The scman driver routes received Ethernet frames as DL_UNITDATA_IND messages up all open and bound streams having a SAP matching the Ethernet type. 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.

PRIMITIVES

In addition to the mandatory connectionless DLPI message set, the scman driver supports the following 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 other SAP and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the six-octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive is not supported.

**scman DRIVER**

The scman driver operates at 10 Mbps, full-duplex.

**PARAMETER LIST**

The scman driver allows you to set and get various parameters for the SUNW, scman device. The parameter list includes current transceiver status, current link status, interpacket gap, local transceiver capabilities, and link partner capabilities.

**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 Fire 15000 servers</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWscdvr</td>
</tr>
</tbody>
</table>

**FILES**

The scman driver utilizes the following files:

/dev/scman

Special character device

/platform/sun4u/kernel/drv/scman.conf

System-wide default device driver properties

**SEE ALSO**

ndd(1M), netstat(1M), driver.conf(4), dman(7D), eri(7D),

dlpi(7P)
<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
<th>Files</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>scmi2c</td>
<td>scmi2c – Smart Transporter chip device driver</td>
<td>dev/scmi2c</td>
<td>smartcard(1M), smartcard(5), ocf_iscr1(7D)</td>
</tr>
<tr>
<td></td>
<td>The scmi2c Smart Transporter device driver is a kernel-loadable Solaris device driver for the Sun Microsystems Smart Transporter chip that features Sun Smartcard internal reader support using the I2C bus interface.</td>
<td>/platform/sun4u/kernel/drv/sparcv9/scmi2c</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCM Microsystems Smart Transporter chip kernel module</td>
<td></td>
</tr>
</tbody>
</table>

**SEE ALSO**

smartcard(1M), smartcard(5), ocf_iscr1(7D)
sc_nct(7D)

NAME  sc_nct – NetraCT-40/60 system management controller (SMC) I2C nexus driver

SYNOPSIS  type=ddi_pseudo;name=nct-ds80ch11-smc

DESCRIPTION  The sc_nct driver is an I2C nexus driver used specifically for the Dallas Semiconductor DS80CH11 microController on the NetraCT-40/60 platform. The driver communicates with the system controller board, power supply, fan tray, and other devices on the I2C bus.

FILES  SUNW,UltraSPARC-IIe-NetraCT-40 32 bit ELF kernel module
        SUNW,UltraSPARC-IIe-NetraCT-60 32 bit ELF kernel module
        /kernel/drv/sparcv9/sc_nct 64 bit 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 (NetraCT series only)</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcti2x.u</td>
</tr>
</tbody>
</table>

SEE ALSO  Writing Device Drivers
NAME  
scsa2usb – SCSI to USB bridge driver

SYNOPSIS  
storage@unit-address

DESCRIPTION  
The scsa2usb driver is a USBA (Solaris USB architecture) compliant nexus driver that supports the USB Bulk Only Mass Storage Specification 1.0. It supports bus-powered and self-powered USB mass storage devices. This nexus driver is a USB client driver. The scsa2usb driver only supports disk devices that utilize the Bulk Only protocol and the SCSI sub class.

The scsa2usb nexus driver maps SCSA target driver requests to USBA client driver requests.

The scsa2usb driver creates a child device info node for each logical unit (LUN) on the mass storage device. The standard Solaris SCSI disk driver is attached to those nodes. Refer to sd(7D).

This driver supports multiple LUN devices and creates a separate child device info node for each LUN. All child LUN nodes attach to sd(7D).

The mass storage device can be managed by rmformat(1). With or without Volume Manager, you can mount, eject, hot remove and hot insert a mass storage device, as the following sections explain.

Mass storage devices are managed by Volume Manager. vold(1M) creates a device nickname which can be listed with eject(1). The device is mounted using volrmount(1) under /rmdisk/label.

See volrmount(1) to unmount the device and eject(1) to eject the media. If the device is ejected while it is mounted, vold(1M) unmounts the device before ejecting it. It also kills any active applications that are accessing the device.

Hot removing a mass storage device with vold(1M) will fail with a console warning. To hot remove or insert a USB storage device first stop vold(1M) by issuing the command /etc/init.d/volmgt stop. After the device has been removed or inserted, restart vold(1M) by issuing the command /etc/init.d/volmgt start.

You can also permanently disable vold for removable devices by commenting out the rmscsi line in vold.conf. See the System Administration Guide, Volume I and Solaris Common Desktop Environment: User’s Guide for details on how to manage a removable device with CDE and Removable Media Manager. See dtfile(1M) under CDE for using Removable Media Manager.

Using mount(1M) and umount(1M)  

Use mount(1M) to mount the device and umount(1M) to unmount the device. Use eject(1) to eject the media. No vold nicknames can be used.

Removing the storage device while it is being accessed or mounted will fail with a console warning. To hot remove the storage device from the system, unmount the file system, then kill all applications accessing the device. Next, hot remove the device. A storage device can be hot inserted at any time.
The following (non-bootable) USB mass-storage devices are compatible with this driver:

<table>
<thead>
<tr>
<th>DEVICE NAME</th>
<th>MEDIUM CAPACITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iomega Zip USB 100/Zip USB 250</td>
<td>100MB/250MB Zip disks</td>
</tr>
<tr>
<td>Iomega Jaz 1GB/2GB drive with Jaz USB adapter</td>
<td>1GB/2GB Jaz disks</td>
</tr>
<tr>
<td>Iomega USB Clik! PC Card Doc</td>
<td>40MB Clik! disks</td>
</tr>
<tr>
<td>Iomega USB Zip CD CD-RW</td>
<td>CD-R, CD-RW media</td>
</tr>
<tr>
<td>Castlewood ORB 2.2GB external USB drive</td>
<td>2.2GB ORB disks</td>
</tr>
<tr>
<td>Hagiwara Sys-Com FlashGate (SmartMedia Reader/Writer USB version)</td>
<td>2MB/4MB (5V) 2MB/4MB/8MB/16MB/32MB/64MB (3.3V)</td>
</tr>
<tr>
<td>Hagiwara Sys-Com FlashGate CF (CompactFlash Reader/Writer USB Version)</td>
<td>8/16/32/48/64/96/128 MB in 3.3V and 5V systems</td>
</tr>
<tr>
<td>SCM Microsystems Inc. SCSI to USB Converter</td>
<td>N/A</td>
</tr>
<tr>
<td>Sony Spressa USB Plus CRX 100E/X2 (CD-RW)</td>
<td>CD-R, CD-RW media</td>
</tr>
</tbody>
</table>

Block special file names are located in /dev/dsk; raw file names are located in /dev/rdsk. Input/output requests to the devices must follow the same restrictions as those for SCSI disks. Refer to sd(7D).

**DEVICE SPECIAL FILES**

- Block files: /dev/dsk/cntn d nsn
- Raw files: /dev/rdsk/cnt n nsn
- Symbolic link to the character device for the media in Zip drive 0: /vol/dev/aliases/zip0
- Symbolic link to the character device for the media in Jaz drive 0: /vol/dev/aliases/jaz0

**IOCTLS**

Refer to dkio(7I) and cdio(7I).

**ERRORS**

Refer to sd(7D).

**FILES**

The device special files for the USB mass storage device are created like those for a SCSI disk. Refer to sd(7D).
/vol/dev/aliases/rmdisk0
Symbolic link to the character device for the media in removable drive 0. This is a
generic removable media device

/kern/drv/scsa2usb
32-bit ELF kernel module

/kern/drv/sparcv9/scsa2usb
64-bit 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>PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

**SEE ALSO**

cfgadm_scsi(1M), dtfile(1M), eject(1), fdisk(1M), mount(1M), rmformat(1),
umount(1M), void(1M), volrmount(1), scsi(4), ohci(7D), sd(7D), uhci(7D),
usba(7D), usb_mid(7D), pcfs(7FS), dkio(7I)

Writing Device Drivers

System Administration Guide, Volume I

Solaris Common Desktop Environment: User’s Guide

Universal Serial Bus Specification 1.1

Universal Serial Bus Mass Storage Class Specification Overview 1.0

Universal Serial Bus Mass Storage Class Bulk-Only Transport 1.0

System Administration Guide: Basic Administration

**DIAGNOSTICS**

Refer to sd(7D).

In addition to being logged, the following messages may appear on the system
console. All messages are formatted in the following manner:

Warning: <device path> (scsa2usb<instance number>): Error Message...

Cannot access device. Please reconnect <name>.

There was an error in accessing the mass-storage device during reconnect. Please
reconnect the device.

Device reported incorrect luns (adjusting to 1).
The mass-storage device reported that it supports invalid number of LUNs. The driver has adjusted the number of LUNs supported to 1.

Device reported <number> luns (adjusting to 1).

The mass-storage device reported that it supports invalid number of LUNs. The driver has adjusted the number of LUNs supported to 1.

Device is busy and cannot be suspended.
Please close files, unmount and eject.

The system wide suspend failed because the mass-storage device is busy. Close the device, unmount the file system and eject the media before retrying the suspend.

Device is not identical to the previous one on this port.
Please disconnect and reconnect.

Another USB device has been inserted on the port that housed a mass-storage device. Please disconnect the USB device and reconnect the mass-storage device back into its port.

Disconnected device was busy, please reconnect.

Disconnection of the mass-storage device failed because the device is busy. Please reconnect the device.

Reinserted device is accessible again.

The mass-storage device that was hot-removed from its USB slot has been re-inserted again to the same slot. It is available for access.

Syncing not supported.

System panic. A file system is mounted on the mass-storage media. Syncing is not supported by scsa2usb driver.

The Zip 100 drive does not comply with Universal Serial Bus Specification 1.0 and cannot be power managed. Power Management support for Zip 100 has been disabled.

If the system panics while a UFS file system is mounted on the mass storage media, no syncing will take place for the mass-storage device. (Syncing is not supported by the scsa2usb driver.) As a result, the file system on the media will not be consistent on reboot.

If a PCFS file system is mounted, no syncing is needed and the filesystem will be consistent on reboot.

If a mass-storage device is busy, system suspend cannot proceed and the system will immediately resume again.

NOTES
Attempts to remove a mass-storage device from the system will fail. The failure will be logged to the console. An attempt to replace the removed device with some other USB device will also fail. To successfully remove a USB mass-storage device you must "close" all references to it.

An Iomega Zip 100Mb disk cannot be formatted on an Iomega Zip250 drive. See the Iomega web site at http://www.iomega.com for details.

Concurrent I/O to devices with multiple LUNs on the same device is not supported.
The `sd` driver supports embedded SCSI-2 and CCS-compatible SCSI disk and CD-ROM drives, ATAPI 2.6 (SFF-8020i)-compliant CD-ROM drives, SFF-8090-compliant SCSI/ATAPI DVD-ROM drives, IOMEGA SCSI/ATAPI ZIP drives, and SCSI JAZ drives. The `sd` driver also supports the Emulex MD21 disk controller for ESDI drives, although support for the MD21 controller may be phased out in subsequent releases.

To determine the disk drive type, use the SCSI/ATAPI inquiry command and read the volume label stored on block 0 of the drive. (The volume label describes the disk geometry and partitioning and must be present for the disk to be mounted by the system.) A volume label is not required for removable, rewritable or read-only media.

The `sd` driver supports embedded SCSI-2 and CCS-compatible SCSI disk and CD-ROM drives, ATAPI 2.6 (SFF-8020i)-compliant CD-ROM drives, SFF-8090-compliant SCSI/ATAPI DVD-ROM drives, IOMEGA SCSI/ATAPI ZIP drives, and SCSI JAZ drives.

The IA BIOS legacy requires a master boot record (MBR) and `fdisk` table in the first physical sector of the bootable media. If the IA hard disk contains a Solaris disk label, it is located in the second 512-byte sector of the FDISK partition.

Block-files access the disk using normal buffering mechanism and are read-from and written-to without regard to physical disk records. A "raw" interface enables direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in a single I/O operation; raw I/O is therefore more efficient when many bytes are transmitted. Block file names are found in `/dev/dsk`; raw file names are found in `/dev/rdsk`.

I/O requests to the raw device must be aligned on a 512-byte (DEVS_BSIZE) boundary and all I/O request lengths must be in multiples of 512 bytes. Requests that do not meet these requirements will trigger an EINVAL error. There are no alignment or length restrictions on I/O requests to the block device.

A CD-ROM disk is single-sided and contains approximately 640 megabytes of data or 74 minutes of audio. When the CD-ROM is opened, the eject button is disabled to prevent manual removal of the disk until the last close() is called. No volume label is required for a CD-ROM. The disk geometry and partitioning information are constant and never change. If the CD-ROM contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

DVD-ROM media can be single or double-sided and can be recorded upon using a single or double layer structure. Double-layer media provides parallel or opposite...
track paths. A DVD-ROM can hold from between 4.5 Gbytes and 17 Gbytes of data, depending on the layer structure used for recording and if the DVD-ROM is single or double-sided.

When the DVD-ROM is opened, the eject button is disabled to prevent the manual removal of a disk until the last close() is called. No volume label is required for a DVD-ROM. If the DVD-ROM contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

ZIP/JAZ media provide varied data capacity points; a single JAZ drive can store up to 2 GBytes of data, while a ZIP-250 can store up to 250MBytes of data. ZIP/JAZ drives can be read-from or written-to using the appropriate drive.

When a ZIP/JAZ drive is opened, the eject button is disabled to prevent the manual removal of a disk until the last close() is called. No volume label is required for a ZIP/JAZ drive. If the ZIP/JAZ drive contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

Each device maintains I/O statistics for the device and for partitions allocated for that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also initiates hi-resolution time stamps at queue entry and exit points to enable monitoring of residence time and cumulative residence-length product for each queue.

Not all device drivers make per-partition IO statistics available for reporting. sd and ssd(7D) per-partition statistics are enabled by default but may disabled in their configuration files.

Refer to dkio(7I), and cdio(7I)

EACCES    Permission denied
EBUSY      The partition was opened exclusively by another thread
EFAULT     The argument features a bad address
EINVAL     Invalid argument. EIO. An I/O error occurred. Refer to notes for details on copy–protected DVD-ROM media
ENOTTY     The device does not support the requested ioctl function
ENXIO      During opening, the device did not exist. During close, the drive unlock failed
EROFS      The device is read-only

The sd driver can be configured by defining properties in the sd.conf file. The sd driver supports the following properties:

enable-partition-kstats
The default value is 1, which causes partition IO statistics to be maintained. Set this value to zero to prevent the driver from recording partition statistics. This slightly
reduces the CPU overhead for IO, minimizes the amount of sar(1) data collected and makes these statistics unavailable for reporting by iostat(1M) even though the -p/-P option is specified. Regardless of this setting, disk IO statistics are always maintained.

qfull-retries
The supplied value is passed as the qfull-retries capability value of the HBA driver. See scsi_ifsetcap(9F) for details.

qfull-retry-interval
The supplied value is passed as the qfull-retry interval capability value of the HBA driver. See scsi_ifsetcap(9F) for details.

allow-bus-device-reset
The default value is 1, which allows resetting to occur. Set this value to 0 (zero) to prevent the sd driver from calling scsi_reset(9F) with a second argument of RESET_TARGET when in error-recovery mode. This scsi_reset(9F) call may prompt the HBA driver to send a SCSI Bus Device Reset message. The scsi_reset(9F) call with a second argument of RESET_TARGET may result from an explicit request via the USCSICMD ioctl. Some high-availability multi-initiator systems may wish to prohibit the Bus Device Reset message; to do this, set the allow-bus-device-reset property to 0.

FILES
sd.conf
driver configuration file
/dev/dsk/ctntndnsn
block files
/dev/rdsk/ctntndnsn
raw files

Where:

<table>
<thead>
<tr>
<th>cn</th>
<th>controller n</th>
</tr>
</thead>
<tbody>
<tr>
<td>tn</td>
<td>SCSI target id n (0-6)</td>
</tr>
<tr>
<td>dn</td>
<td>SCSI LUN n (0-7 normally; some HBAs support LUNs to 15 or 32. See the specific manpage for details)</td>
</tr>
<tr>
<td>sn</td>
<td>partition n (0-7)</td>
</tr>
</tbody>
</table>

IA Only /dev/rdsk/ctntndn/pn
raw files

Where:

| pn  | Where n=0 the node corresponds to the entire disk. |

SEE ALSO
sar(1), fdisk(1M), format(1M), iostat(1M), close(2), ioctl(2), lseek(2), read(2), write(2), driver.conf(4), scsi(4), filesystem(5), pcfs(7FS), hsfs(7FS), cdio(7I), dkio(7I), scsi_ifsetcap(9F), scsi_reset(9F)

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

Emulex MD21 Disk Controller Programmer Reference Manual

Device and Network Interfaces 461
ATA Packet Interface for CD-ROMs, SFF-8020i

Mt.Fuji Commands for CD and DVD, SFF8090v3

DIAGNOSTICS

Error for Command: `<command name>`
Error Level: Fatal
Requested Block: `<n>`
Error Block: `<m>`
Vendor: `<vendorname>`
Serial Number: `<serial number>`
Sense Key: `<sense key name>`
ASC: 0x<a> (<ASC name>), ASCQ: 0x<b>, FRU: 0x<c>

The command indicated by `<command name>` failed. The Requested Block is the block where the transfer started and the Error Block is the block that caused the error. Sense Key, ASC, and ASCQ information is returned by the target in response to a request sense command.

Caddy not inserted in drive

The drive is not ready because no caddy has been inserted.

Check Condition on REQUEST SENSE

A REQUEST SENSE command completed with a check condition. The original command will be retried a number of times.

Label says `<m>` blocks Drive says `<n>` blocks

There is a discrepancy between the label and what the drive returned on the READ CAPACITY command.

Not enough sense information

The request sense data was less than expected.

Request Sense couldn’t get sense data

The REQUEST SENSE command did not transfer any data.

Reservation Conflict

The drive was reserved by another initiator.

SCSI transport failed: reason ‘xxxx’: {retrying|giving up}

The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

Unhandled Sense Key

The REQUEST SENSE data included an invalid sense.

Unit not ready. Additional sense code 0x
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

The disk label is corrupted.

No resources for dumping

A packet could not be allocated during dumping.

Offline

Drive went offline; probably powered down.

Requeue of command fails

Driver attempted to retry a command and experienced a transport error.

sdrestart transport failed()

Driver attempted to retry a command and experienced a transport error.

Transfer length not modulo

Illegal request size.

Transport of request sense fails()

Driver attempted to submit a request sense command and failed.

Transport rejected()

Host adapter driver was unable to accept a command.

Unable to read label

Failure to read disk label.

Unit does not respond to selection

Drive went offline; probably powered down.

DVD-ROM media containing DVD-Video data may follow/adhere to the requirements of content scrambling system or copy protection scheme. Reading of copy-protected sector will cause I/O error. Users are advised to use the appropriate playback software to view video contents on DVD-ROM media containing DVD-Video data.
se(7D)

NAME
se – Siemens 82532 ESCC serial communications driver

SYNOPSIS
se @bus_address: port_name [, cu]

DESCRIPTION
The se module is a loadable STREAMS driver that provides basic support for the 82532 ESCC hardware and basic asynchronous and synchronous communication support. This manual page describes the asynchronous protocol interface; for information on the synchronous interface, please see the se_hdlc(7D) manual page.

The platform specific device bus address for the se module is bus_address. The se module's port_name is a single letter (a-z).

The Siemens 82532 provides two serial input/output channels capable of supporting a variety of communication protocols. A typical system will use one of these devices to implement two serial ports (port_name), usually configured for RS-423 (which also supports most RS-232 equipment). The Siemens 82532 uses 64 character input and output FIFOs to reduce system overhead. When receiving characters, the CPU is notified when 32 characters have arrived (one-half of receive buffer is full) or no character has arrived in the time it would take to receive four characters at the current baud rate.

When sending characters, the Siemens 82532 places the first 64 characters to be sent into its output FIFO and then notifies the CPU when it is half empty (32 characters left). Because the se module waits for the Siemens 82532 to transmit the remaining characters within its output FIFO before making requested changes, delays may occur when the port's attributes are being modified.

The se module implements CTS/RTS flow control in hardware. To prevent data overruns, remove CTS/RTS flow control responsibility from the CPU during periods of high system load.

In async mode (obtained by opening /dev/cua/[a-z], /dev/term/[a-z] or /dev/tty[a-z]), the driver supports the termio(7I) device control functions specified by flags in the c_cflag word of the termios structure, and by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the c_iflag word. All other termio(7I) functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the ldterm(7M) and ttcompat(7M) STREAMS modules are automatically pushed on top of the stream, providing the standard termio interface.

Each of the following are valid name space entries: /dev/cua/[a-z], /dev/term/[a-z], and /dev/tty[a-z]. The number of entries used in this name space are machine dependent. The /dev/tty[a-z] device names exist only if the SunOS 4.x Binary Compatibility Package is installed. The /dev/tty[a-z] device names are created by the ucblinks command, which is available only with the SunOS 4.x Binary Compatibility Package.

You can connect a single tty line to a modem for incoming and outgoing calls using a special feature controlled by the minor device number. By accessing character-special
devices with names of the form /dev/cua/[a-z], it is possible to open a port without the Carrier Detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

After a /dev/cua/[a-z] line is opened, the corresponding tty line cannot be opened until the /dev/cua/[a-z] line is closed. A blocking open will wait until the /dev/cua/[a-z] line is closed (which will drop Data Terminal Ready and Carrier Detect) and carrier is detected again. A non-blocking open will return an error. If the tty line has been opened successfully (usually only when carrier is recognized on the modem), the corresponding /dev/cua/[a-z] line cannot be opened. This allows a modem to be attached to a device, (for example, /dev/term/[a-z] renamed from /dev/tty[a-z]) and used for dial-in (by enabling the line for login in /etc/inittab) and dial-out (by tip(1) or uucp(1C)) as /dev/cua/[a-z] when no one is logged in on the line.

The se module supports the standard set of termio ioctl() calls.

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

The state of the DCD, CTS, RTS, and DTR interface signals can be queried through the use of the TIOCM_CAR, TIOCM_CTS, TIOCM_RTS, and TIOCM_DTR arguments to the TIOCMGET ioctl command, respectively. Due to hardware limitations, only the RTS and DTR signals may be set through their respective arguments to the TIOCMSET, TIOCMBIS, and TIOCMBIC ioctl commands.

The input and output line speeds may be set to all baud rates supported by termio. Input and output line speeds cannot be set independently; when you set the output speed, the input speed is automatically set to the same speed.

When using baud rates over 100,000 baud, the software changes the line driver configuration to handle the higher data rates. This action decreases the theoretical maximum cable length from 70 meters to 30 meters.

When the se module is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default. A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. Due to the risk of incorrect sequence interpretation, binary protocols such as PPP, SLIP and others should not be run over the serial console port when the Alternate Break sequence is in effect. By default, the Alternate Break sequence is a three character sequence: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For information on breaking (entering the debugger or monitor), see kadb(1) and kb(7M).

An open() will fail under the following conditions:

ENXIO  The unit being opened does not exist.
EBUSY The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.

EBUSY The port is in use by another serial protocol.

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

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

FILES
/dev/cua/[a-z] dial-out tty lines
/dev/term/[a-z] dial-in tty lines
/dev/tty[a-z] binary compatibility package device names
/dev/se_hdlc[0-9] synchronous devices - see se_hdlc(7D).
/dev/se_hdlc synchronous control clone device

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

tip(1), kadb(1), ucblinks(1B), cu(1C), uucp(1C), ports(1M), ioctl(2), open(2), attributes(5), zs(7D), zsh(7D), se_hdlc(7D), termio(7I), ldterm(7M), ttcompat(7M), kb(7M)

SunOS 4.x Binary Compatibility Guide

DIAGNOSTICS

sen : fifo overrun The Siemens 82532 internal FIFO received more data than it could handle. This indicates that Solaris was not servicing data interrupts fast enough and suggests a system with too many interrupts or a data line with a data rate that is too high.

sen : buffer overrun The se module was unable to store data it removed from the Siemens 82532 FIFO. The user process is not reading data fast enough, and suggests an overloaded system. If possible, the application should enable flow control (either CTSRTS or XONXOFF) to allow the driver to backpressure the remote system when the local buffers fill up.
se_hdlc(7D)

<table>
<thead>
<tr>
<th>NAME</th>
<th>se_hdlc – on-board high-performance serial HDLC interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td><code>se@bus_address:port_number[, hdlc]</code></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>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).</td>
</tr>
<tr>
<td>APPLICATION PROGRAMMING INTERFACE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>IOCTLS</td>
<td>The se driver supports the following ioctl() commands.</td>
</tr>
</tbody>
</table>

468 man pages section 7: Device and Network Interfaces • Last Revised 1 Jan 1997
S_IOCGETMODE  
Return a struct scc_mode containing parameters currently in use. These include the transmit and receive clock sources, boolean loopback and NRZI mode flags and the integer baud rate.

S_IOCSETMODE  
The argument is a struct scc_mode from which the ESCC channel will be programmed.

S_IOCGETSTATS  
Return a struct sl_stats containing the current totals of hardware-generated events. These include numbers of packets and characters sent and received by the driver, aborts and CRC errors detected, transmit underruns, and receive overruns.

S_IOCCLRSTATS  
Clear the hardware statistics for this channel.

S_IOCGETSPEED  
Returns the currently set baud rate as an integer. This may not reflect the actual data transfer rate if external clocks are used.

S_IOCGETMCTL  
Returns the current state of the CTS and DCD incoming modem interface signals as an integer.

The following structures are used with se_hdlc ioctl() commands:

```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 under the following conditions:

ENXIO  
The unit being opened does not exist.
EBUSY
The device is in use by another serial protocol.

An ioctl() will fail under the following conditions:

EINVAL
An attempt was made to select an invalid clocking source.

EINVAL
The baud rate specified for use with the baud rate generator would translate to a null time constant in the ESCC’s registers.

/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=mnn
   A kernel memory allocation failed for one of the private data structures. The value of mnn 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.

sen 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  

   ses@target, lun

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  

The following is an example of the ses.conf file format:

    #
    # Copyright (c) 1996, by Sun Microsystems, Inc.
    # All rights reserved.
    #
    #ident "@(#)ses.conf 1.1 97/02/10 SMI"
    #

    name="ses" parent="sf"
    target=15;

    name="ses" parent="SUNW,pln" port=0 target=15;
    name="ses" parent="SUNW,pln" port=1 target=15;
    name="ses" parent="SUNW,pln" port=2 target=15;
    name="ses" parent="SUNW,pln" port=3 target=15;
    name="ses" parent="SUNW,pln" port=4 target=15;
    name="ses" parent="SUNW,pln" port=5 target=15;

    name="ses" class="scsi"
    target=15 lun=0;

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)
NAME | sesio – enclosure services device driver interface  
SYNOPSIS | `#include<sys/sesio.h>`  
DESCRIPTION | The `ses` device driver provides the following ioctls as a means to access SCSI enclosure services devices.  
IOCTLS | The `ses` driver supports the following ioctls:  
|  | SES_IOCTL_GETSTATE | This ioctl obtains enclosure state in the `ses_ioctl` structure.  
|  | SES_IOCTL_SETSTATE | This ioctl is used to set parameters on the enclosure services device. The `ses_ioctl` structure is used to pass information into the driver.  
ERRORS | EIO | The `ses` driver was unable to obtain data from the enclosure services device or the data transfer could not be completed.  
|  | ENOTTY | The `ses` driver does not support the requested ioctl function.  
|  | ENXIO | The enclosure services device does not exist.  
|  | EFAULT | The user specified a bad data length.  
STRUCTURES | The `ses_ioctl` structure has the following fields:  
|  | uint32_t; /* Size of buffer that follows */  
|  | uint8_t page_code; /* Page to be read/written */  
|  | uint8_t reserved[3]; /* Reserved; set to 0 */  
|  | unit8t buffer[1]; /* Size arbitrary, user specifies */  
EXAMPLES | **EXAMPLE 1** Using the SES_IOCTL_GETSTATE ioctl  
The following example uses the SES_IOCTL_GETSTATE ioctl to recover 20 bytes of page 4 from a previously opened device.  
|  | char abuf[30];  
|  | struct ses_ioctl *sesp;  
|  | int status;  
|  | sesp = (ses_ioctl *)abuf;  
|  | sesp->size = 20;  
|  | sesp->page_code = 4;  
|  | status = ioctl(fd, SES_IOCTL_GETSTATE, abuf);  
ATTRIBUTES | See attributes(5) for descriptions of the following attributes:  
|  | ATTRIBUTE TYPE | ATTRIBUTE VALUE |  
|  | Architecture | SPARC |  
SEE ALSO | `ses(7D), ioctl(9E)`  

472 man pages section 7: Device and Network Interfaces • Last Revised 27 Mar 1997
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)` covers 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.

```
/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**

`luxadm(1M), prtconf(1M), driver.conf(4), socal(7D), ssd(7D)`

**DEVICE AND NETWORK INTERFACES**

*ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)*

*ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP)*

*ANSI X3.270-1996, SCSI-3 Architecture Model (SAM)*

*Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)*

**DIAGNOSTICS**

In addition to being logged, the messages below may display on the system console.

The first set of messages indicate that the attachment was unsuccessful, and will only display while the `sf` driver is initially attempting to attach. Each message is preceded by `sf%d`, where `%d` is the instance number of the `sf` device.
Failed to alloc soft state
    Driver was unable to allocate space for the internal state structure. Driver did not
    attach to device, SCSI devices will be inaccessible.

Bad soft state
    Driver requested an invalid internal state structure. Driver did not attach to device,
    SCSI devices will be inaccessible.

Failed to obtain transport handle
    Driver was unable to obtain a transport handle to communicate with the socal
driver. Driver did not attach to device, SCSI devices will be inaccessible.

Failed to allocate command/response pool
    Driver was unable to allocate space for commands and responses. Driver did not
    attach to device, SCSI devices will be inaccessible.

Failed to allocate kmem cache
    Driver was unable to allocate space for the packet cache. Driver did not attach
to device, SCSI devices will be inaccessible.

Failed to allocate dma handle for
    Driver was unable to allocate a dma handle for the loop map. Driver did not attach
to device, SCSI devices will be inaccessible.

Failed to allocate lilp map
    Driver was unable to allocate space for the loop map. Driver did not attach
to device, SCSI devices will be inaccessible.

Failed to bind dma handle for
    Driver was unable to bind a dma handle for the loop map. Driver did not attach
to device, SCSI devices will be inaccessible.

Failed to attach
    Driver was unable to attach for some reason that may be printed. Driver did not
    attach to device, SCSI devices will be inaccessible.

The next set of messages may display at any time. The full device pathname, followed
by the shorter form described above, will precede the message.

Invalid lilp map
    The driver did not obtain a valid lilp map from the socal driver. SCSI device will be
    inaccessible.

Target t, AL-PA x and hard
    The device with a switch setting t has an AL-PA x which does not match its hard
    address y. The device will not be accessible.

Duplicate switch settings
    The driver detected devices with the same switch setting. All such devices will be
    inaccessible.

WWN changed on target t
    The World Wide Name (WWN) has changed on the device with switch setting t.
Target t, unknown device type
The driver does not know the device type reported by the device with switch setting t.
The `sgen` driver exports the `uscsi(7I)` interfaces to user processes. The `sgen` driver can be configured to bind to SCSI devices for which no system driver is available. Examples of such devices include SCSI scanners and SCSI processor devices.

Typically, drivers which export the `uscsi(7I)` interface unconditionally require that the user present superuser credentials. The `sgen` driver does not, and relies on the file system permissions on its device special file to govern who may access that device. By default, access is restricted and device nodes created by the `sgen` driver are readable and writable by the superuser exclusively.

It is important to understand that SCSI devices coexisting on the same SCSI bus may potentially interact with each other. This may result from firmware bugs in SCSI devices, or may be made to happen programatically by sending appropriate SCSI commands to a device. Potentially, any application controlling a device via the `sgen` driver can introduce data integrity or security problems in that device or any other device sharing the same SCSI bus.

Granting unprivileged users access to an `sgen`-controlled SCSI device may create other problems. It may be possible for a user to instruct a target device to gather data from another target device on the same bus. It may also be possible for malicious users to install new firmware onto a device to which they are granted access. For environments where security is a concern, but user access to devices controlled by the `sgen` driver is nevertheless desired, it is recommended that the devices be separated onto a dedicated SCSI bus to mitigate the risk of data corruption and security violations.

The `sgen` driver is configurable via the `sgen.conf` file. In addition to standard SCSI device configuration directives (see `scsi(4)`), administrators can set several additional properties for the `sgen` driver.

By default, the `sgen` driver will not claim or bind to any devices on the system. To do so, it must be configured by the administrator using the `inquiry-config-list` and/or the `device-type-config-list` properties.

As with other SCSI drivers, the `sgen.conf` configuration file enumerates the targets `sgen` should use. See `scsi(4)` for more details. For each target enumerated in the `sgen.conf` file, the `sgen` driver sends a SCSI `INQUIRY` command to gather information about the device present at that target. The `inquiry-config-list` property specifies that the `sgen` driver should bind to a particular device returning a particular set of inquiry data. The `device-type-config-list` specifies that the `sgen` driver should bind to every device that is of a particular SCSI device type. When examining the device, the `sgen` driver tests to see if it matches an entry in the
When a match against the INQUIRY data presented by a device is made, the sgen driver attaches to that device and creates a device node and link in the /devices and /dev hierarchies. See the FILES section for more information about how these files are named.

It is important for the administrator to ensure that devices claimed by the sgen driver do not conflict with existing target drivers on the system. For example, if the sgen driver is configured to bind to a direct access device, the standard sd.conf file will usually cause sd to claim the device as well. This can cause unpredictable results. In general, the uscsi(7I) interface exported by sd(7D) or st(7D) should be used to gain access to direct access and sequential devices.

The sgen driver is disabled by default. The sgen.conf file is shipped with all of the 'name="sgen" class="scsi" target=...' entries commented out to shorten boot time and to prevent the driver from consuming kernel resources. To use the sgen driver effectively on desktop systems, simply uncomment all of the name="sgen" lines in sgen.conf file. On larger systems with many SCSI controllers, carefully edit the sgen.conf file so that sgen binds only where needed. Refer to driver.conf(4) for further details.

inquiry-config-list

The inquiry-config-list property is a list of pairs of strings; it enumerates a list of specific devices to which the sgen driver will bind. Each pair of strings is referred to as <vendorid, productid> in the discussion below.

vendorid

is used to match the Vendor ID reported by the device. The SCSI specification limits Vendor IDs to eight characters. Correspondingly, the length of this string should not exceed eight characters. As a special case, "*" may be used as a wildcard which matches any Vendor ID. This is useful in situations where more than one vendor produces a particular model of a product. vendorid is matched against the Vendor ID reported by the device in a case-insensitive manner.

productid

is used to match the Product ID reported by the device. The SCSI specification limits Product IDs to sixteen characters (unused characters are filled with the whitespace characters). Correspondingly, the length of productid should not exceed sixteen characters. When examining the Product ID of the device, sgen examines the length l of productid and performs a match against only the first l characters in the device’s Product ID. productid is matched against the Product ID reported by the device in a case-insensitive manner.

For example, to match some fictitious devices from ACME corp, the inquiry-config-list can be configured as follows:
inquiry-config-list = "ACME", "UltraToast 3000",
                  "ACME" "UltraToast 4000",
                  "ACME", "UltraToast 5000";

To match "UltraToast 4000" devices, regardless of vendor, inquiry-config-list is modified as follows:

inquiry-config-list = "*", "UltraToast 4000";

To match every device from ACME in the "UltraToast" series (i.e. UltraToast 3000, 4000, 5000,...), inquiry-config-list is modified as follows:

inquiry-config-list = "ACME" "UltraToast";

Whitespace characters are significant when specifying productid. For example, a productid of "UltraToast 1000" is fifteen characters in length. If a device reported its ID as "UltraToast 10000", the sgen driver would bind to it because only the first fifteen characters are considered significant when matching. To remedy this situation, specify productid as "UltraToast 1000 ", (note trailing space). This forces the sgen driver to consider all sixteen characters in the product ID to be significant.

device-type-config-list

The device-type-config-list property is a list of strings; it enumerates a list of device types to which the sgen driver will bind. The valid device types correspond to those defined by the SCSI-S PDC Draft Standard, Rev. 11a. These types are:

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Inquiry Type ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct</td>
<td>0x00</td>
</tr>
<tr>
<td>sequential</td>
<td>0x01</td>
</tr>
<tr>
<td>printer</td>
<td>0x02</td>
</tr>
<tr>
<td>processor</td>
<td>0x03</td>
</tr>
<tr>
<td>worm</td>
<td>0x04</td>
</tr>
<tr>
<td>rodirect</td>
<td>0x05</td>
</tr>
<tr>
<td>scanner</td>
<td>0x06</td>
</tr>
<tr>
<td>optical</td>
<td>0x07</td>
</tr>
</tbody>
</table>
Alternately, you can specify device types by `INQUIRY` type ID. To do this, specify `type_0x<typenum>` in the `sgen-config-list`. Case is not significant when specifying device type names.

`sgen-diag` property sets the diagnostic output level. This property can be set globally and/or per target/lun pair. `sgen-diag` is an integer property, and can be set to 0, 1, 2 or 3. Illegal values will silently default to 0. The meaning of each diagnostic level is as follows:

0  No error reporting [default]
1  Report driver configuration information, unusual conditions, and indicate when sense data has been returned from the device.
2  Trace the entry into and exit from routines inside the driver, and provide extended diagnostic data. No error reporting [default].
3  Provide detailed output about command characteristics, driver state, and the contents of each CDB passed to the driver.

In ascending order, each level includes the diagnostics that the previous level reports. See the IOCTLS section for more information on the `SGEN_IOC_DIAG` ioctl.

**FILES**

`sgen.conf`

Driver configuration file. See CONFIGURATION for more details.

```
/dev/scsi<devtype>/cntn<dn>
```

The `sgen` driver categorizes each device in a separate directory by its SCSI device type. The files inside the directory are named according to their controller number, target ID and LUN as follows:

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Inquiry Type ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>changer</td>
<td>0x08</td>
</tr>
<tr>
<td>comm</td>
<td>0x09</td>
</tr>
<tr>
<td>prepress1</td>
<td>0x0a</td>
</tr>
<tr>
<td>prepress2</td>
<td>0x0b</td>
</tr>
<tr>
<td>array_ctrl</td>
<td>0x0c</td>
</tr>
<tr>
<td>ses</td>
<td>0x0d</td>
</tr>
<tr>
<td>rbc</td>
<td>0x0e</td>
</tr>
<tr>
<td>ocrw</td>
<td>0x0f</td>
</tr>
<tr>
<td>bridge</td>
<td>0x10</td>
</tr>
<tr>
<td>type_unknown</td>
<td>0x1f</td>
</tr>
</tbody>
</table>
This is analogous to the \{controller;target;device\} naming scheme, and the
controller numbers correspond to the same controller numbers which are used for
naming disks. For example, /dev/dsk/c0t0d0s0 and
/dev/scsi/scanner/c0t5d0 are both connected to controller c0.

The \texttt{sgen} driver exports the \texttt{uscsi(7I)} interface for each device it manages. This
allows a user process to talk directly to a SCSI device for which there is no other driver
installed in the system. Additionally, the \texttt{sgen} driver supports the following ioctls:

\begin{itemize}
\item \texttt{SGEN_IOC_READY}
  Send a \texttt{TEST UNIT READY} command to the device and return 0 upon success,
  non-zero upon failure. This ioctl accepts no arguments.
\item \texttt{SGEN_IOC_DIAG}
  Change the level of diagnostic reporting provided by the driver. This ioctl accepts a
  single integer argument between 0 and 3. The levels have the same meaning as in
  the \texttt{sgen-diag} property discussed in PROPERTIES above.
\end{itemize}

\texttt{EBUSY}
\quad The device was opened by another thread or process. The driver
\quad maintains a strict exclusive-open policy for each device.

\texttt{ENXIO}
\quad During opening, the device did not respond to a \texttt{TEST UNIT
\quad READY} SCSI command.

\texttt{ENOTTY}
\quad Indicates that the device does not support the requested ioctl
\quad function.

Here is an example of how \texttt{sgen} can be configured to bind to scanner devices on the
system:

\begin{verbatim}
device-type-config-list = "scanner";
\end{verbatim}

The administrator should subsequently uncomment the appropriate
name="sgen"... lines for the SCSI target ID to which the scanner corresponds. In
this example, the scanner is at target 4.

\begin{verbatim}
name = "sgen" class= "scsi" target=4 lun=0;
\end{verbatim}

If it is expected that the scanner will be moved from target to target over time, or that
more scanners might be added in the future, it is recommended that all of the
name="sgen"... lines be uncommented, so that \texttt{sgen} checks all of the targets on the
bus.

For large systems where boot times are a concern, it is recommended that the
parent="" property be used to specify which SCSI bus \texttt{sgen} should examine.

\textbf{SEE ALSO}
\texttt{driver.conf(4), scsi(4), sd(7D), st(7D), uscsi(7I)}
Writing Device Drivers

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

SCSI-3 SPC Draft Standard, Rev. 11a
## NAME
sk98sol – SysKonnect Gigabit Ethernet SK-98xx device driver

## SYNOPSIS
```
/dev/skge
/kernel/drv/sk98sol
```

## DESCRIPTION
The sk98sol driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface (DLPI), over a SysKonnect Gigabit Ethernet adapter (SK-98xx series). The driver supports multiple installed SysKonnect SK-98xx adapters. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

The driver provides the `/dev/skge` cloning character-special device as well as per-adapter character-special devices `/dev/skgex`, where `x` represents the device instance number.

The sk98sol driver is a Style 1 and Style 2 Data Link Service (DLS) provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. See `dlpi(7P)`.

An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened Stream with a particular device (`ppa`). This is unnecessary and invalid for DLPI Style 1. The `ppa` ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for the system.

The device is initialized on first attach and de-initialized (stopped) upon last detach. Valid device numbers for all detected adapters are displayed on the console at driver startup time and are written to the `/var/adm/messages` log file.

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

- Maximum SDU is 1500 (9000 if JumboFrames are enabled).
- Minimum SDU is 0.
- DLSAP address length is 8 bytes.
- MAC type is `DL_CSMACD`.
- SAP length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Service mode is `DL_CLDLS`.
- Optional quality of service (QOS) support is not included; as a result, the QOS field values are 0.
- Provider style is `DL_STYLE2`.
- Version is `DL_VERSION_2`.

---

482 man pages section 7: Device and Network Interfaces • Last Revised 10 Oct 2000
Parameters are set in the `/kernel/drv/sk98sol.conf` configuration file, which is created during installation. See `driver.conf(4)`. You can edit the `/kernel/drv/sk98sol.conf` file to reflect your settings and reboot the system to use the new parameter values. If the file exists prior to driver installation, the new parameter values will be used as soon as the driver is installed.

String parameter values must be surrounded with double quotes ("), while integer parameter values are not. Parameter names and values are case sensitive and you should use them exactly as shown.

The parameters discussed in this section can be set for each port on the adapter.

In each of the following descriptions, ? represents port A or B.

**AutoNegotiation_?**

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>On, Off, Sense</td>
</tr>
<tr>
<td>Default</td>
<td>Sense (for SK-984x fiber adapters).</td>
</tr>
<tr>
<td>Default</td>
<td>On (for SK-982x copper adapters.)</td>
</tr>
</tbody>
</table>

The Sense value automatically detects whether the link partner supports autonegotiation. If your link partner is configured to half duplex with autonegotiation turned off, set the `AutoNegotiation_?` and `DuplexCapabilities_?` parameters manually. Do not set the `AutoNegotiation_?` parameter value to Sense, as it will fail.

Do not use Sense for 1000Base-T (copper) adapters. If Sense is selected, it will be mapped to On automatically.

**DuplexCapabilities_?**

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Half, Full, Both</td>
</tr>
<tr>
<td>Default</td>
<td>Both</td>
</tr>
</tbody>
</table>

Set the `DuplexCapabilities_?` parameter only if the `AutoNegotiation_?` parameter is set to the On or Off values. If `AutoNegotiation_?` is set to On, all three `AutoNegotiation_?` values are possible; however, if set to Off, only the Full and Half values are allowed.

Use the `DuplexCapabilities_?` parameter if your link partner does not support all possible combinations.
**FlowControl_?**

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Sym, SymOrRem, LocSend, None</td>
</tr>
<tr>
<td>Default</td>
<td>SymOrRem</td>
</tr>
</tbody>
</table>

Use the `FlowControl_?` parameter to set the flow control capabilities reported by the port during autonegotiation:

**Sym**

Symmetric flow control, where both link partners are allowed to send PAUSE frames.

**SymOrRem**

SymmetricOrRemote flow control, where both link partners or only the remote partner are allowed to send PAUSE frames.

**LocSend**

LocalSend flow control, where only the local link partner is allowed to send PAUSE frames.

**None**

No flow control, where no link partner is allowed to send PAUSE frames.

The `FlowControl_?` parameter is ignored if `AutoNegotiation_?` is set to "off."

**Role_?**

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Auto, Master, Slave</td>
</tr>
<tr>
<td>Default</td>
<td>Auto</td>
</tr>
</tbody>
</table>

Use the `Role_?` parameter only for the SK-9821 and SK-9822 adapters.

1000Base-T communication between two ports requires one port to act as the master (and provide timing information) and the other as slave. Normally, this is negotiated between the two ports during link establishment. If this fails, use the `Role_?` parameter to force the master and slave roles on the ports. If `AutoNegotiation_?` is set to "off," then the `Role_?` parameter must be set manually.

**PreferredPort**

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>A, B</td>
</tr>
<tr>
<td>Default</td>
<td>A</td>
</tr>
</tbody>
</table>
Use the *PreferredPort* parameter to force the preferred port to A or B (on two-port NICs). The preferred port is the port selected if both ports are detected as fully functional.

**RlmtMode**

<table>
<thead>
<tr>
<th>Type:</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values:</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Default:</td>
<td>1</td>
</tr>
</tbody>
</table>

RLMT (Redundant Link Management Technology) provides three modes to determine if a port is available for use.

1. Check link state only: use the link state reported by the adapter hardware for each individual port.
2. Check other port: RLMT sends test frames from one port to another and checks if they are received. The ports must be connected to the network that allow LLC test frames to be exchanged (that is, networks without routers between the ports).
3. Check other port and segmentation: RLMT checks the other port and also requests information from the Gigabit Ethernet switch next to each port to determine if the network is segmented between the ports. Only use this mode if you have Gigabit Ethernet switches installed and configured to use the Spanning Tree protocol.

Note that modes 2 and 3 are meant to operate in configurations where a network path exists between the ports on a single adapter. They are **not** designed to work in networks where adapters are connected back-to-back.

**JumboFrames**

<table>
<thead>
<tr>
<th>Type:</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values:</td>
<td>Off, On</td>
</tr>
<tr>
<td>Default:</td>
<td>Off</td>
</tr>
</tbody>
</table>

To enable support for *JumboFrames* (frames with a length of up to 9014 bytes), set *JumboFrames* to "On." Because longer frames reduce operating system overhead, *JumboFrames* increases network throughput.

For full *JumboFrames* support, the maximum transfer unit (MTU) size used by TCP/IP must also be changed by using the `ifconfig(1M)` command. To do this, remove the comment sign (#) before the `ifconfig` line in the `/etc/rcS.d/S50sk98sol` file. You should also ensure that the adapter device number (*skge0*) matches the attach
number displayed during system startup. The MTU must be set to 9000, not including
the 14 bytes of MAC address header.

JumboFrames can only be used if all equipment in your subnetwork supports them;
currently many switches do not support JumboFrames. Devices without Jumbo
Support will drop the longer frames (and might report them as error frames). If you
experience problems with this, connect two SK-98xx adapters (with JumboFrames
enabled) back-to-back.)

CopyThreshold

<table>
<thead>
<tr>
<th>Type:</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values:</td>
<td>0-1500</td>
</tr>
<tr>
<td>Default:</td>
<td>1500</td>
</tr>
</tbody>
</table>

During transmit, the driver relies on the frame’s physical memory address to tell the
hardware where to find the frame data. Setting up the DMA address can take time on
Solaris; it may be more convenient to copy the frame data to a buffer that you have set
up in advance. All frames with a length less than or equal to the CopyThreshold
parameter value are copied into buffers; for longer frames, the real DMA setup is
done. By default (without JumboFrames support), all frames are copied. You can
experiment with this parameter to find out if your system performs better with only
smaller frames copied.

To use more complex syntax for setting different parameters on multiple adapters, see
driver.conf(4). For example:

```bash
name="sk98sol" parent="/pci@1f,4000" unit-address="2"
AutoNegotiation_A="Off"
name="sk98sol" parent="/pci@1f,2000" unit-address="2"
AutoNegotiation_B="Sense"
```

**DIAGNOSTICS**

If multiple NICs are installed in the system, the following message may appear on the
console and in the /var/adm/messages log file:

Allocation of descriptor memory failed

You can avoid this message by tuning the lomempages kernel parameter. By default,
the value of this parameter is 36 pages. Each SK-98xx adapter requires a determined
number of pages, so increase the value of the lomempages parameter in increments of
ten pages until all NICs in the system run correctly.

To modify the value of this parameter to 46 pages, append the set lomempages=46
line to the /etc/system file and reboot the system.
FILES

/dev/skge  Character special device
/dev/skgor  Per-adapter character special device, where x is the adapter ppa
/kernel/drv/sk98sol  ELF kernel module
/kernel/drv/sparcv9/sk98sol  ELF kernel module (64-bit SPARC version)
/kernel/drv/sk98sol.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>IA, SPARC</td>
</tr>
</tbody>
</table>

SEE ALSO

ifconfig(1M), netstat(1M), driver.conf(4), attributes(5), dlpi(7P).

sk98sol.txt driver README file — Included in the driver package; also available from www.syskonnect.com.
The skfp FDDI driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface (DLPI) over a SysKonnect FDDI PCI adapter. The driver supports multiple installed SysKonnect FDDI PCI adapters. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The skfp driver supports all SysKonnect SK-NET FDDI PCI adapters (SK-55xx (32-bit) and SK-58xx (64-bit) series) on 32-bit systems, and the SK-58xx series on 64-bit systems.

The skfp driver provides the /dev/skfp cloning character-special device that accesses all SK-NET FDDI PCI adapters using Data Link Service (DLS) Style 2. It also provides per-adapter character-special devices /dev/skfpX, (where X represents the device instance number) that access a special NIC using DLS Style 1.

The skfp driver is a Style 1 and Style 2 DLS provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in <sys/dlpi.h>. See dlpi(7P).

An explicit DL_ATTACH_REQ message by the user is required to associate the opened Stream with a particular device (ppa). This is unnecessary and invalid for DLPI Style 1. The ppa ID is interpreted as an unsigned long data type and indicates the corresponding device instance (unit) number. An error (DL_ERROR_ACK) is returned by the driver if the ppa field value does not correspond to a valid device instance number for the system.

The device is initialized on first attach and de-initialized (stopped) upon last detach. Valid device numbers for all detected adapters are displayed on the console at driver startup time and written to the /var/adm/messages log file.

The values returned in the DL_INFO_ACK primitive in response to the DL_INFO_REQ request are:

- Maximum SDU is 4470.
- Minimum SDU is 0.
- DSLAP address length is 8 bytes.
- MAC type is DL_FDDI.
- SAP length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Service mode is DL_CLDLS.
- Optional quality of service (QOS) support is not included; as a result, the QOS field values are 0.
- Provider style is DL_STYLE2.
- Version is DL_VERSION_2.

**OPTIONS**
Options are not required for normal operation. In special cases, FDDI Station Management (SMT) parameters can be modified by using the `/usr/bin/smtpara` utility (see the driver README files). The `smtpara` utility should be used only by those very familiar with FDDI.

**DIAGNOSTICS**
If multiple NICs are installed in the system, the following message may appear on the console and in the `/var/adm/messages` log file:

```
skfp: DMA memory allocation failed!
```

You can avoid this message by tuning the `lomempages` kernel parameter. By default, the value of this parameter is 36 pages. Each SK-FDDI PCI adapter requires nine pages, so increase the value of the `lomempages` parameter in increments of nine pages until all NICs in the system run correctly.

To modify the value of this parameter to 45 pages, you can, for example, append the `set lomempages=45` line to the `/etc/system` file and reboot the system.

**FILES**
- `/dev/skfp` Character special device
- `/dev/skfp` Per-adapter character special device, where `x` is the adapter `ppa`
- `/kernel/drv/skfp` ELF kernel module
- `/kernel/drv/sparcv9/skfp` ELF kernel module (64-bit SPARC version)
- `/kernel/drv/skfp.conf` Driver configuration file
- `/usr/bin/smtpara` SMT parameter utility
- `/etc/fddi.cfg` smtpara configuration file
- `<sys/dlpi.h>` DLPI definitions

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

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

**SEE ALSO**
- `ifconfig(1M)`, `netstat(1M)`, `attributes(5)`, `dlpi(7P)`
- `skfp.txt` (32-bit driver) and `skfpx.txt` (64-bit driver) README files — Included in the driver package or available from [www.syskonnect.com](http://www.syskonnect.com).
The Service Location Protocol ("SLP") is a dynamic service discovery protocol that runs on top of the Internet Protocol ("IP"). The protocol is specified by the IETF standard-track documents RFC 2165, RFC 2608, RFC 2609; the API is documented in RFC 2614.

There are two components to the SLP technology. The first is a daemon, `slpd(1M)`, which coordinates SLP operations. The second is a software library, `slp_api(3slp)`, through which processes access a public API. Both components are configured by means of the SLP configuration file, `slp.conf(4)`.

The SLP API is useful for two types of processes:

- **Client Applications**
  Services and service information can be requested from the API. Clients do not need to know the location of a required service, only the type of service, and optionally, the service characteristics. SLP will supply the location and other information to the client through the API.

- **Server Processes**
  Programs that offer network services use the SLP API to advertise their location as well as other service information. The advertisement can optionally include attributes describing the service. Advertisements are accompanied by a lifetime; when the lifetime expires, the advertisement is flushed, unless it is refreshed prior to expiration.

API libraries are available for both the C and Java languages.

SLP provides the following additional features:

- `slpd(1M)` can be configured to function as a transparent directory agent. This feature makes SLP scalable to the enterprise. System administrators can configure directory agents to achieve a number of different strategies for scalability.

- SLP service advertising and discovery is performed in scopes. Unless otherwise configured, all discovery and all advertisements are in the scope `default`. In the case of a larger network, scopes can be used to group services and client systems so that users will only find those services which are physically near them, belong to their department, or satisfy the specified criteria. Administrators can configure these scopes to achieve different service provider strategies.

- Services may be registered by proxy through a serialized registration file. This is an alternative to registering services through the API. See `slpd.reg(4)` for more information.

### Attributes

See `attributes(5)` for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWslpu</td>
</tr>
<tr>
<td>CSI</td>
<td>CSI-enabled</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Standard</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

slpd(1m), slp_api(3slp), slp.conf(4), slpd.reg(4), attributes(5)


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 user file systems or system reserved areas.

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

ATTRIBUTE TYPE | ATTRIBUTE VALUE
--- | ---
Architecture | IA

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

SEE ALSO smart2cfg(1), fdisk(1M), attributes(5), cmdk(7D)
NAME  soc – Serial Optical Controller (SOC) device driver

SYNOPSIS  soc@<sbus-slot>, <a>

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 may appear on system console in addition to being logged.

On the console, these messages are preceded by:

```
socal%d: port %a
```

where %d is the instance number of the socal controller and %a is the port on the host adapter.

Fibre Channel Loop is ONLINE

The Fibre Channel loop is now online.

Fibre Channel Loop is OFFLINE

The Fibre Channel loop is now offline.

attach failed: device in slave-only slot.

Move soc+ card to another slot.

attach failed: bad soft state.

Driver did not attach, devices will be inaccessible.

attach failed: unable to alloc xport struct.

Driver did not attach, devices will be inaccessible.
attach failed: unable to map eeprom
   Driver was unable to map device memory; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: unable to map XRAM
   Driver was unable to map device memory; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: unable to map registers
   Driver was unable to map device registers; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: unable to access status register
   Driver was unable to map device registers; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: unable to install interrupt handler
   Driver was not able to add the interrupt routine to the kernel. Driver did not attach
   to device, devices will be inaccessible.

attach failed: unable to access host adapter XRAM
   Driver was unable to access device RAM; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: unable to write host adapter XRAM
   Driver was unable to write device RAM; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.

attach failed: read/write mismatch in XRAM
   Driver was unable to verify device RAM; check for bad hardware. Driver did not
   attach to device, devices will be inaccessible.
NAME
sockio – ioctlts that operate directly on sockets

SYNOPSIS
#include <sys/sockio.h>

DESCRIPTION
The ioctlts listed in this manual page apply directly to sockets, independent of any underlying protocol. The setsockopt() call (see getsockopt(3SOCKET)) is the primary method for operating on sockets, rather than on the underlying protocol or network interface. ioctlts for a specific network interface or protocol are documented in the manual page for that interface or protocol.

<table>
<thead>
<tr>
<th>ioctl</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIOCSPGRP</td>
<td>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).</td>
</tr>
<tr>
<td>SIOCGPGRP</td>
<td>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.</td>
</tr>
<tr>
<td>SIOCCATMARK</td>
<td>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.</td>
</tr>
</tbody>
</table>

SEE ALSO
ioctl(2), getsockopt(3SOCKET)
spwr(7D)

NAME    spwr – SMC EtherPower II 10/100 (9432) Ethernet device driver

SYNOPSIS /dev/spwr

DESCRIPTION The spwr Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over SMC EtherPower II 10/100 controllers. Multiple EtherPower II controllers installed within the system are supported by the driver. The spwr driver provides basic support for the SMC EtherPower II hardware. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting.

APPLICATION The cloning character-special device /dev/spwr is used to access all SMC EtherPower II devices installed within the system.

PROGRAMMING The spwr driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the spwr driver with the DLPI and STREAMS functionality required of a LAN driver. See gld(7d) for more details on the primitives supported by the driver.

INTERFACE The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are as follows:

- The maximum SDU is 1500 (ETHERMTU).
- The minimum SDU is 0. The spwr driver will pad to the mandatory 60-octet minimum packet size.
- The DLSAP address length is 8.
- The MAC type is DL_ETHER.
- The SAP length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- The broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

FILES /dev/spwr
Character special device.
/kernel/drv/spwr.conf
Driver configuration file

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

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

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

### DEVICE STATISTICS SUPPORT
Each device maintains I/O statistics for the device and for partitions allocated for that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also initiates hi-resolution time stamps at queue entry and exit points to enable monitoring of residence time and cumulative residence-length product for each queue.

Not all device drivers make per-partition I/O statistics available for reporting. `ssd` and `sd(7D)` per-partition statistics are enabled by default but may disabled in their configuration files.

### IOCTLS
Refer to `dkio(7I)`.

### ERRORS
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Permission denied.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>The partition was opened exclusively by another thread.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The argument was a bad address.</td>
</tr>
</tbody>
</table>
EINVAL  Invalid argument.
EIO     An I/O error occurred.
ENOTTY  The device does not support the requested ioctl function.
ENXIO   When returned during open(2), this error indicates the device
does not exist.
EROFS  The device is a read-only device.

CONFIGURATION
The ssd driver can be configured by defining properties in the ssd.conf file. The
ssd driver supports the following properties:

enable-partition-kstats
The default value is 1, which causes partition IO statistics to be maintained. Set this
value to zero to prevent the driver from recording partition statistics. This slightly
reduces the CPU overhead for IO, minimizes the amount of sar(1) data collected
and makes these statistics unavailable for reporting by iostat(1M) even though
the -p/-P option is specified. Regardless of this setting, disk IO statistics are
always maintained.

FILES
ssd.conf     driver configuration file
/dev/dsk/cnt nans n    block files
/dev/rdsk/cnt nans n    raw files

where, for the SPARCstorage Array:

cnt  is the controller number on the system. Each SPARCstorage Array will
      have a unique controller number

tn   port number within the SPARCstorage Array n

dn   SCSI target n

sn   partition n

and for all FC-AL disks:

cnt  is the controller number on the system.

tn   7-bit disk loop identifier, such as switch setting

dn   SCSI lun n

sn   partition n (0-7)

SEE ALSO sar(1), format(1M), iostat(1M), ioctl(2), lseek(2), open(2), read(2), write(2),
driver.conf(4), cdio(7I), dkio(7I)

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

SPARCstorage Array User’s Guide
Error for command '<command name>' Error Level: Fatal Requested Block <n>, Error Block: <m>, Sense Key: <sense key name>, Vendor '<vendor name>': ASC = 0x<a> (<ASC name>), ASCQ = 0x<b>, FRU = 0x<c>

The command indicated by <command name> failed. The Requested Block is the block where the transfer started and the Error Block is the block that caused the error. Sense Key, ASC, and ASCQ information is returned by the target in response to a request sense command.

Check Condition on REQUEST SENSE

A REQUEST SENSE command completed with a check condition. The original command will be retried a number of times.

Label says <m> blocks Drive says <n> blocks

There is a discrepancy between the label and what the drive returned on the READ CAPACITY command.

Not enough sense information

The request sense data was less than expected.

Request Sense couldn’t get sense data

The REQUEST SENSE command did not transfer any data.

Reservation Conflict

The drive was reserved by another initiator.

SCSI transport failed: reason ‘xxxx’: [retrying|giving up]

The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

Unhandled Sense Key <n>

The REQUEST SENSE data included an invalid sense key.

Unit not Ready. Additional sense code 0x<n>

The drive is not ready.

corrupt label - bad geometry

The disk label is corrupted.

corrupt label - label checksum failed

The disk label is corrupted.
The disk label is corrupted.

The drive returned busy during a number of retries.

The drive was probably powered down or died.

The geometry of the drive could not be established.

There was a residue after the command completed normally.

The drive is not ready.

A bp with consistent memory could not be allocated.

Free memory pool exhausted.

Free memory pool exhausted.

A packet could not be allocated during dumping.

Drive went offline; probably powered down.

Driver attempted to retry a command and experienced a transport error.

Driver attempted to retry a command and experienced a transport error.

Illegal request size.

transport rejected

ssd(7D)

502 man pages section 7: Device and Network Interfaces • Last Revised 21 Feb 2001
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**

```bash
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**

The st driver now supports persistent errors (see mtio(7I)) and asynchronous tape operations (see mtio(7I), aioread(3AIO), and aiowrite(3AIO)).

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).

**Read 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.

**Write Operation**

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>;

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.

<prettprint> 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT_ISQIC</td>
<td>0x32</td>
</tr>
<tr>
<td>MT_ISREEL</td>
<td>0x33</td>
</tr>
<tr>
<td>MT_ISDAT</td>
<td>0x34</td>
</tr>
<tr>
<td>MT_IS8MM</td>
<td>0x35</td>
</tr>
<tr>
<td>MT_ISOTHER</td>
<td>0x36</td>
</tr>
</tbody>
</table>

<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>
<tr>
<td>ST_EJECT_TAPE_ON_CHANGER_FAILURE</td>
<td>0x200000</td>
</tr>
<tr>
<td>ST_RETRY_ON_RECOVERED_DEFERRED_ERROR</td>
<td>0x400000</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.

**ST_KNONS_EOD**
If flag is set, the device can determine when EOD (End of Data) has been reached. When this flag is set, the `st` driver uses fast file skipping. Otherwise, file skipping happens one file at a time.

**ST_UNLOADABLE**
The flag indicates the device will not complain if the `st` driver is unloaded and loaded again (see `modload(1M)` and `modunload(1M)`). That is, the driver will return the correct inquiry string.

**ST_SOFT_ERROR_REPORTING**
The flag indicates the tape device will perform a “request sense” or “log sense” command when the device is closed. Currently, only Exabyte and DAT drives support this feature.

**ST_LONG_TIMEOUTS**
The flag indicates the tape device requires timeouts that are 5 times longer than usual for normal operation.

**ST_BUFFERED_WRITES**
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.

**ST_NO_REC_SIZE_LIMIT (SPARC Only)**
The flag applies to variable-length tape devices. If this flag is set, the record size is not limited to a 64 Kbyte record size. The record size is only limited by the smaller of either the record size supported by the device or the maximum DMA transfer size of the system. (Refer to Large Record Sizes and WARNINGS.)
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.

The ST_NO_RESERVE_RELEASE flag disables the use of reserve on open and release on close. If an attempt to use an ioctl of MTRESERVE or MTRERELEASE on a drive with this flag set, it will return an error of ENOTTY (inappropriate ioctl for device).

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 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.

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.
If ST_EJECT_TAPE_ON_CHANGER_FAILURE flag is set, the tape will be ejected automatically if the tape cartridge is trapped in the medium due to positioning problems of the medium changer.

The following ASC/ASCQ keys are defined to the reasons for causing tape ejection if ST_EJECT_TAPE_ON_CHANGER_FAILURE option is set to 0x200000:

<table>
<thead>
<tr>
<th>Sense ASC/ASCQ Description</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 15/01 Mechanical Failure</td>
<td>4</td>
</tr>
<tr>
<td>4 44/00 Internal Target Failure</td>
<td>4</td>
</tr>
<tr>
<td>2 53/00 Media Load or Eject Failed</td>
<td>2</td>
</tr>
<tr>
<td>4 53/00 Media Load or Eject Failed</td>
<td>4</td>
</tr>
<tr>
<td>4 53/01 Unload Tape Failure</td>
<td>4</td>
</tr>
</tbody>
</table>

If ST_RETRY_ON_RECOVERED_DEFERRED_ERROR flag is set, the st driver will retry the last write if this cmd caused a check condition with error code 0x71 and sense code 0x01. Some tape drives, notably the IBM 3090, require this option.

<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

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 Global tape-config list property**

The following is an example of a global tape-config-list property:

```bash
tape-config-list =
  "Magic DAT", "Magic 4mm Helical Scan", "magic-data";
```
### EXAMPLE 1 Global tape-config list property

(Continued)

```plaintext
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;
  ...
name="st" class="scsi"
  target=6 lun=0;
```

### EXAMPLE 2 Tape-config-list property applicable to target 2 only

The following is an example of a tape-config-list property applicable to target 2 only:

```plaintext
name="st" class="scsi"
  target=0 lun=0;
name="st" class="scsi"
  target=1 lun=0;
name="st" class="scsi"
  target=2 lun=0
  tape-config-list =
    "Magic DAT", "Magic 4mm Helical Scan", "magic-data"
  magic-data = 1,0x34,1024,0x1639,4,0,0x8c,0x8c,0x8c,3;
name="st" class="scsi"
  target=3 lun=0;
  ...
name="st" class="scsi"
  target=6 lun=0;
```

### Large Record Sizes

To support applications such as seismic programs that require large record sizes, the flag `ST_NO_RECSIZE_LIMIT` must be set in drive option in the configuration entry. A SCSI tape drive that needs to transfer large records should OR this flag with other flags in the ‘options’ field in `st.conf`. (Refer to Tape Configuration.) By default, this flag is set for the built-in config entries of Archive DAT and Exabyte drives.

If this flag is set, the `st` driver issues a `SCSI-2 READ BLOCK LIMITS` command to the device to determine the maximum record size allowed by it. If the command fails, `st` continues to use the maximum record sizes mentioned in the `mtio(7I)` man page.

If the command succeeds, `st` restricts the maximum transfer size of a variable-length device to the minimum of that record size and the maximum DMA size that the host adapter can handle. Fixed-length devices are bound by the maximum DMA size.
allocated by the machine. Note that tapes created with a large record size may not be readable by earlier releases or on other platforms.

(Refer to the WARNINGS section for more information.)

**EOT Handling**

The Emulex drives have only a physical end of tape (PEOT); thus it is not possible to write past EOT. All other drives have a logical end of tape (LEOT) before PEOT to guarantee flushing the data onto the tape. The amount of storage between LEOT and PEOT varies from less than 1 Mbyte to about 20 Mbyte, depending on the tape drive.

If EOT is encountered while writing an Emulex, no error is reported but the number of bytes transferred is 0 and no further writing is allowed. On all other drives, the first write that encounters EOT will return a short count or 0. If a short count is returned, then the next write will return 0. After a zero count is returned, the next write returns a full count or short count. A following write returns 0 again. It is important that the number and size of trailer records be kept as small as possible to prevent data loss. Therefore, writing after EOT is not recommended.

Reading past EOT is transparent to the user. Reading is stopped only by reading EOF’s. For 1/2” reel devices, it is possible to read off the end of the reel if one reads past the two file marks which mark the end of recorded media.

**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="scsi"
  target=0 lun=0;
name="st" class="scsi"
  target=1 lun=0;
name="st" class="scsi"
  target=2 lun=0;
....
```

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 `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.

**FILES**

- `/kernel/drv/st.conf`
  - driver configuration file
- `/usr/include/sys/mtio.h`
  - structures and definitions for mag tape io control commands
- `/usr/include/sys/scsi/targets/stdef.h`
  - definitions for SCSI tape drives
- `/dev/rmt/ [0–127] [l,m,h,u,c] [b] [n]`
  - where `l,m,h,u,c` specifies the density (low, medium, high, ultra/compressed), `b` the optional BSD behavior (see `mtio(7I)`), and `n` the optional no rewind behavior. For example, `/dev/rmt/0lb` specifies unit 0, low density, BSD behavior, and no rewind.

For 1/2” reel tape devices (HP-88780), the densities are:

<table>
<thead>
<tr>
<th>Density</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>800 BPI density</td>
</tr>
<tr>
<td>m</td>
<td>1600 BPI density</td>
</tr>
<tr>
<td>h</td>
<td>6250 BPI density</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>c</td>
<td>data compression</td>
</tr>
<tr>
<td></td>
<td>(not supported on all modules)</td>
</tr>
</tbody>
</table>

For 8mm tape devices (Exabyte 8200/8500/8505):

<table>
<thead>
<tr>
<th>l</th>
<th>Standard 2 Gbyte format</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>5 Gbyte format (8500, 8505 only)</td>
</tr>
<tr>
<td>h, c</td>
<td>5 Gbyte compressed format (8505 only)</td>
</tr>
</tbody>
</table>

For 4mm DAT tape devices (Archive Python):

<table>
<thead>
<tr>
<th>l</th>
<th>Standard format</th>
</tr>
</thead>
<tbody>
<tr>
<td>m, h, c</td>
<td>data compression</td>
</tr>
</tbody>
</table>

For all QIC (other than QIC-24) tape devices:

<table>
<thead>
<tr>
<th>l, m, h, c</th>
<th>density of the tape cartridge type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(not all devices can read and write all formats)</td>
</tr>
</tbody>
</table>

For QIC-24 tape devices (Emulex MT-02):

<table>
<thead>
<tr>
<th>l</th>
<th>QIC-11 Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>m, h, c</td>
<td>QIC-24 Format</td>
</tr>
</tbody>
</table>

SEE ALSO

mt(1), modload(1M), modunload(1M), open(2), read(2), write(2), aioread(3AIO), aiowrite(3AIO), kstat(3KSTAT), driver.conf(4), scsi(4), standards(5), esp(7D), isp(7D), mtio(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>, PRU = 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.

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.)

Tape devices that do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for tape devices supplied by Sun Microsystems.

Tape devices that do not report a blank check error at the end of recorded media may cause file positioning operations to fail. Some tape drives, for example, mistakenly report media error instead of blank check error.
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 eight 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, and 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 termio(7I) device control functions specified by flags in the c_cflag word of the termios(3C) structure. In addition, the serial ports support the IGNPAR, PARMRK, INPCK, IXON, IXANY, and IXOFF flags in the c_iflag word of the termios(3C) 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(3C) (c_iflag) parameters apply to it. Trying to execute a nonsensical ioctl() on the parallel port is not recommended.

All other termios(3C) functions are performed by STREAMS modules pushed on top of 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/stclp 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 that indicates which dial-in line it is (so that /dev/term/0 is the first dial-in line). 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
The parallel port is given the name /dev/stclp\text{n}, where \text{n} is the SPC/S unit number (see Minor Numbers, below).

The control port, named /dev/stcn\text{n}, where \text{n} is the SPC/S, is available. An ioctl() is provided for this special file which allow the collection of statistics maintained on serial port performance.

### Minor Numbers

The characters \text{o p u u l l l l} correspond to the bits in the minor number. They are mnemonic indicators of the function of the corresponding bit.

- \text{o} set if this device is an outgoing serial line
- \text{p} set if this is a parallel port device
- \text{u} device unit number
- \text{l} device line number if this is the parallel port line, 'p' should be 1 and 'll' 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 is supported by the stc driver on both the serial and parallel ports.

If the CRTSCTS flag in the \text{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(71).

### DEVICE-SPECIFIC IOCTLs

The stc driver supports two additional ioctl()s. STC_SPPC(struct ppc_params_t *) sets parallel port parameters, and STC_GPPC(struct ppc_params_t *) gets parallel port parameters. Both are valid until changed or until a close().

```c
struct ppc_params_t {
    uint_t flags; /* driver status flag */
}```
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. STC_GSTATS(struct stc_stats_t *) gets or resets driver performance statistics on serial ports.
uint_t reserved; /* this field is meaningless */

The **STC_GSTATS ioctl**() works only on the SPC/S control port. The possible **cmd** values, defined in `/usr/include/sys/stcio.h`, are **STAT_CLEAR**, which clears the line statistics, and **STAT_GET**, which gets the line statistics.

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_pn**, 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 dtr_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 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.

rxfifo-  The UART receive fifo threshold.

For serial ports, the value-carrying tokens have the following defaults and ranges:

<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 minus 2 bytes</td>
</tr>
<tr>
<td>drain_size</td>
<td>64 bytes</td>
<td>4 bytes</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>rtpr</td>
<td>18 millisecs</td>
<td>1 millisec</td>
<td>255 millisecs</td>
</tr>
<tr>
<td>rxfifo</td>
<td>4 bytes</td>
<td>1 byte</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.
- **busy_timeout**: The amount of time in seconds to wait for a BUSY signal to clear, or zero for an infinite BUSY timeout.
- **data_setup**: The amount of time in microseconds between placing data on the parallel lines and asserting the STROBE.
- **strobe_width**: Width of the STROBE pulse, in microseconds.

For value-carrying tokens for parallel ports:

<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 microseconds</td>
<td>1 microsec</td>
<td>30 microsec</td>
</tr>
<tr>
<td>data_setup</td>
<td>2 microseconds</td>
<td>0 microsec</td>
<td>30 microsec</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>
</tbody>
</table>
### 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 its SELECT line until after the next character is sent to the printer. Since the stc driver will not send data to the device if its 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_SPPCioctl() call.

### 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   The stc driver uses the following 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(3C), ldterm(7M), termio(7I), termiox(7I),
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. Fatal errors usually occur during the modload process.

ERROR    Some condition has disrupted the normal operation of the board and/or device driver. There may be data loss. This class of message may indicate an impending hardware failure.

ADVISORY The device driver has detected a condition that may be of interest, usually a transient condition that clears itself.

The following messages can be generated during initialization of the driver or board.

stc_attach: can’t allocate memory for unit structs
   FATAL: kmem_zalloc() failed to allocate memory for the driver’s internal data structures.
stcattach: board revision undeterminable
   FATAL. The driver did not get a hardware revision level from the board’s onboard FCode PROM.

stcattach: board revision 0x%x not supported by driver.
   FATAL. This revision of the board is not supported by the driver.

stcattach: oscillator revision undeterminable
   FATAL. The driver did not get an oscillator revision level from the board’s onboard FCode PROM.

stcattach: 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.

stcattach: error initializing stc%d
   FATAL. An error occurred while trying to initialize the board; perhaps a memory access failed.

stcattach: bad number of interrupts: %d
   FATAL. An incorrect number of interrupts was read from the board’s onboard FCode PROM.

stcattach: 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 0xff, was: 0x%x
   FATAL. Either 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 0xff, was: 0x%x
   FATAL. Either 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, V1.0 (0x4) and V1.1 (0x5), have been released. 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. An SPC/S card in a system running Solaris 2.X. requires a V1.1 PROM.

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 correctly map the SBus interrupt levels that the SPC/S uses.

524  man pages section 7: Device and Network Interfaces • Last Revised 17 Mar 1998

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
Messages Related
To The Parallel
Port

stc(7D)
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.

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 may be picking up induced noise, causing the ppc to generate an unwanted interrupt; or this could indicate an internal problem in the ppc.

ppc_acktimeout: unit %d can’t get pointer to read q
  ERROR. The driver’s internal ppc data structure became corrupted.

ppc_acktimeout: unit %d can’t send M_ERROR message
  ERROR. The driver can’t send an M_ERROR STREAMS message to the application.

ppc_signal: unit %d can’t get pointer to read q
  ERROR. The driver’s internal ppc data structure became corrupted.

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).

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, which should only happen if an application explicitly sends this message.

stc_wput: unit %d line %d unknown message: 0x%x
  ADVISORY. An unknown STREAMS message was sent to the driver. Check your application coding.

stc_start: unit %d line %d unknown message: 0x%x
  ADVISORY. An unknown STREAMS message was sent to the driver. Check your application coding.

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(9F) 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_vcmd: unit %d cd-180 firmware revision: 0x%x
  ADVISORY. This message displays the firmware revision level of the cd-180 when
  the driver is first loaded.
### NAME
stp4020

### 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 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(3)) ioctl commands are a subset of the ioctl(2) commands
              and perform a variety of control functions on streams.

              The fildes argument is an open file descriptor that refers to a stream. The command
              argument determines the control function to be performed as described below. The arg
              argument represents additional information that is needed by this command. The type
              of arg depends upon the command, but it is generally an integer or a pointer to a
              command-specific data structure. The command and arg arguments are interpreted by
              the STREAM head. Certain combinations of these arguments may be passed to a
              module or driver in the stream.

              Since these STREAMS commands are ioctl's, they are subject to the errors described
              in ioctl(2). In addition to those errors, the call will fail with errno set to EINVAL,
              without processing a control function, if the STREAM referenced by fildes is linked
              below a multiplexor, or if command is not a valid value for a stream.

              Also, as described in ioctl(2), STREAMS modules and drivers can detect errors. In
              this case, the module or driver sends an error message to the STREAM head
              containing an error value. This causes subsequent calls to fail with errno set to this
              value.

IOCTLS  The following ioctl commands, with error values indicated, are applicable to all
              STREAMS files:

              I_PUSH  Pushes the module whose name is pointed to by arg onto the top
                      of the current stream, just below the STREAM head. If the
                      STREAM is a pipe, the module will be inserted between the stream
                      heads of both ends of the pipe. It then calls the open routine of the
                      newly-pushed module. On failure, errno is set to one of the
                      following values:
                      EINVAL  Invalid module name.
                      EFAULT  arg points outside the allocated address space.
                      ENXIO   Open routine of new module failed.
                     ENXIO   Hangup received on fildes.

              I_POP  Removes the module just below the STREAM head of the
                      STREAM pointed to by fildes. To remove a module from a pipe
                      requires that the module was pushed on the side it is being
                      removed from. arg should be 0 in an I_POP request. On failure,
                      errno is set to one of the following values:
EINVAL  No module present in the stream.
ENXIO   Hangup received on fildes.
EPERM   Attempt to pop through an anchor by an unprivileged process.

I_ANCHOR Positions the stream anchor to be at the STREAMS module directly below the STREAM head. Once this has been done, only a privileged process may pop modules below the anchor on the stream. arg must be 0 in an I_ANCHOR request. On failure, errno is set to the following value:
EINVAL  Request to put an anchor on a pipe.

I_LOOK Retrieves the name of the module just below the STREAM head of the STREAM pointed to by fildes, and places it in a null terminated character string pointed at by arg. The buffer pointed to by arg should be at least FMNAMESZ+1 bytes long. This requires the declaration #include <sys/conf.h>. On failure, errno is set to one of the following values:
EFAULT  arg points outside the allocated address space.
EINVAL  No module present in stream.

I_FLUSH This request flushes all input and/or output queues, depending on the value of arg. Legal arg values are:
FLUSHR  Flush read queues.
FLUSHW  Flush write queues.
FLUSHRW Flush read and write queues.

If a pipe or FIFO does not have any modules pushed, the read queue of the STREAM head on either end is flushed depending on the value of arg.
If FLUSHR is set and fildes is a pipe, the read queue for that end of the pipe is flushed and the write queue for the other end is flushed. If fildes is a FIFO, both queues are flushed.
If FLUSHW is set and fildes is a pipe and the other end of the pipe exists, the read queue for the other end of the pipe is flushed and the write queue for this end is flushed. If fildes is a FIFO, both queues of the FIFO are flushed.
If FLUSHRW is set, all read queues are flushed, that is, the read queue for the FIFO and the read queue on both ends of the pipe are flushed.
Correct flush handling of a pipe or FIFO with modules pushed is achieved via the `pipemod` module. This module should be the first module pushed onto a pipe so that it is at the midpoint of the pipe itself.

On failure, `errno` is set to one of the following values:

- **ENOSR**  
  Unable to allocate buffers for flush message due to insufficient 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:

EINVAL arg value is invalid or arg is zero and process is not registered to receive the SIGPOLL signal.

EAGAIN Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by arg, where the events are those specified in the description of I_SETSIG above. On failure, errno is set to one of the following values:

EINVAL Process not registered to receive the SIGPOLL signal.
EFAULT

 arg points outside the allocated address space.

I_FIND

Compares the names of all modules currently present in the
STREAM to the name pointed to by arg, and returns 1 if the named
module is present in the stream. It returns 0 if the named module
is not present. On failure, errno is set to one of the following
values:

EFAULT

 arg points outside the allocated address space.

EINVAL

 arg does not contain a valid module name.

I_PEEK

Allows a user to retrieve the information in the first message on
the STREAM head read queue without taking the message off the
queue. I_PEEK is analogous to getmsg(2) except that it does not
remove the message from the queue. arg points to a strpeek
structure, which contains the following members:

struct strbuf ctlbuf;
struct strbuf databuf;
long flags; The maxlen field in the ctlbuf and databuf
strbuf structures (see getmsg(2)) must be set to the number of
bytes of control information and/or data information, respectively,
to retrieve. flags may be set to RS_HIPRI or 0. If RS_HIPRI is
set, I_PEEK will look for a high priority message on the STREAM
head read queue. Otherwise, I_PEEK will look for the first
message on the STREAM head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no
message was found on the STREAM head read queue. It does not
wait for a message to arrive. On return, ctlbuf specifies
information in the control buffer, databuf specifies information in
the data buffer, and flags contains the value RS_HIPRI or 0. On
failure, errno is set to the following value:

EFAULT

 arg points, or the buffer area specified in
ctlbuf or databuf is, outside the allocated
address space.

EBADMSG

Queued message to be read is not valid for
I_PEEK.

EINVAL

 Illegal value for flags.

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`:

- **RPROTNR**M: Reject `read()` with `EBADMSG` if a control message is at the front of the STREAM head read queue.
- **RPROTDA**T: Deliver the control portion of a message as data when a user issues `read()`. This is the default behavior.
- **RPROTD**I: 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:

```c
struct strbuf ctlbuf;
struct strbuf databuf;
t_uscalar_t flags;
int fildes;
int offset;
```

The `len` member in the `ctlbuf strbuf` structure (see `putmsg(2)`) must be set to the size of a `t_uscalar_t` plus the number of bytes of control information to be sent with the
message. The fildes member specifies the file descriptor of the other STREAM, and the offset member, which must be suitably aligned for use as a t_uscalar_t, specifies the offset from the start of the control buffer where I_FDINSERT will store a t_uscalar_t whose interpretation is specific to the STREAM end. The len member in the databuf strbuf structure must be set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.

The flags member specifies the type of message to be created. A normal message is created if flags is set to 0, and a high-priority message is created if flags is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY or O_NONBLOCK is set. Instead, it fails and sets errno to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NDELAY or O_NONBLOCK has been specified. No partial message is sent.

The ioctl() function with the I_FDINSERT command will fail if:

EAGAIN A non-priority message is specified, the O_NDELAY or O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions.

ENOSR Buffers can not be allocated for the message that is to be created.

EFAULT The arg argument points, or the buffer area specified in ctlbuf or databuf is, outside the allocated address space.

EINVAL One of the following: The fildes member of the strfdinsert structure is not a valid, open STREAM file descriptor; the size of a t_uscalar_t plus offset is greater than the len member for the buffer specified through ctlptr; the offset member does not specify a properly-aligned location in the data buffer; or an undefined value is stored in flags.
I_FDINSERT can also fail if an error message was received by the
STREAM head of the STREAM corresponding to the fildes
member of the strfdinsert structure. In this case, errno
will be set to the value in the message.

I_STR
Constructs an internal STREAMS ioctl message from the data
pointed to by arg, and sends that message downstream.

This mechanism is provided to send user ioctl requests to
downstream modules and drivers. It allows information to be sent
with the ioctl, and will return to the user any information sent
upstream by the downstream recipient. I_STR blocks until the
system responds with either a positive or negative
acknowledgement message, or until the request "times out" after
some period of time. If the request times out, it fails with errno
set to ETIME.

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:

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 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 `fdles`.
- **ETIME**: A downstream `ioctl` timed out before acknowledgement was received.

An `I_STR` can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the STREAM head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the `ioctl` command sent downstream fails. For these cases, `I_STR` will fail with `errno` set to the value in the message.

### I_SWROPT

Sets the write mode using the value of the argument `arg`. Legal bit settings for `arg` are:

- **SNDZERO**: Send a zero-length message downstream when a write of 0 bytes occurs.

To not send a zero-length message when a write of 0 bytes occurs, this bit must not be set in `arg`.

On failure, `errno` may be set to the following value:

- **EINVAL**: `arg` is not the above legal value.
I_GWROPT

Returns the current write mode setting, as described above, in the int that is pointed to by the argument arg.

I_SENDFD

Requests the STREAM associated with fildes to send a message, containing a file pointer, to the stream head at the other end of a STREAM pipe. The file pointer corresponds to arg, which must be an open file descriptor.

I_SENDFD converts arg into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see intro(3)) of the STREAM head at the other end of the STREAM pipe to which it is connected. On failure, errno is set to one of the following values:

- EAGAIN: The sending STREAM is unable to allocate a message block to contain the file pointer.
- EAGAIN: The read queue of the receiving STREAM head is full and cannot accept the message sent by I_SENDFD.
- EBADF: arg is not a valid, open file descriptor.
- EINVAL: fildes is not connected to a STREAM pipe.
- ENXIO: Hangup received on fildes.

I_RECVFD

Retrieves the file descriptor associated with the message sent by an I_SENDFD ioctl over a STREAM pipe. arg is a pointer to a data buffer large enough to hold an strrecvfd data structure containing the following members:

```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 an 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.

**I_CKBAND** Check if the message of a given priority band exists on the stream head read queue. This returns 1 if a message of a given priority exists, 0 if not, or −1 on error. **arg** should be an integer containing the value of the priority band in question. On failure, **errno** is set to the following value:

**EINVAL** Invalid arg value.

**I_GETBAND** Returns the priority band of the first message on the STREAM head read queue in the integer referenced by **arg**. On failure, **errno** is set to the following value:

**ENODATA** No message on the STREAM head read queue.

**I_CANPUT** Check if a certain band is writable. **arg** is set to the priority band in question. The return value is 0 if the priority band **arg** is flow controlled, 1 if the band is writable, or −1 on error. On failure, **errno** is set to the following value:

**EINVAL** Invalid arg value.

**I_SETCLTIME** Allows the user to set the time the STREAM head will delay when a stream is closing and there are data on the write queues. Before closing each module and driver, the STREAM head will delay for the specified amount of time to allow the data to drain. Note, however, that the module or driver may itself delay in its close routine; this delay is independent of the STREAM head’s delay and is not settable. If, after the delay, data are still present, data will be flushed. **arg** is the number of milliseconds to delay, rounded up to the nearest legal value on the system. The default is fifteen seconds. On failure, **errno** is set to the following value:

**EINVAL** Invalid arg value.

**I_GETCLTIME** Returns the close time delay in the integer pointed by **arg**.

**I_SERROPT** Sets the error mode using the value of the argument **arg**.

Normally STREAM head errors are persistent; once they are set due to an **M_ERROR** or **M_HANGUP**, the error condition will remain until the STREAM is closed. This option can be used to set the
STREAM head into non-persistent error mode i.e. once the error has been returned in response to a read(2), getmsg(2), ioctl(2), write(2), or putmsg(2) call the error condition will be cleared. The error mode can be controlled independently for read and write side errors. Legal arg values are either none or one of:

- **RERRNORM**: Persistent read errors, the default.
- **RERRNONPERSIST**: Non-persistent read errors.

OR'ed with either none or one of:

- **WERRNORM**: Persistent write errors, the default.
- **WERRNONPERSIST**: Non-persistent write errors.

When no value is specified e.g. for the read side error behavior then the behavior for that side will be left unchanged.

On failure, errno is set to the following value:

- **EINVAL**: arg is not one of the above legal values.

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.

**I_GERROPT**

Returns the current error mode setting in an int pointed to by the argument arg. Error modes are described above for I_SERROPT. On failure, errno is set to the following value:

- **EFAULT**: arg points outside the allocated address space.
Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

arg is not a valid, open file descriptor.

fildes STREAM does not support multiplexing.

arg is not a stream, or is already linked under a multiplexor.

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.

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.

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:

Hangup received on fildes.

Time out before acknowledgement message was received at STREAM head.

Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

arg is an invalid multiplexor ID number or fildes is not the STREAM on which the I_LINK that returned arg was performed.

fildes is the file descriptor of a pipe or FIFO.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the STREAM head of fildes. In addition, an error code can be returned in the positive or negative
acknowledgement message. For these cases, \texttt{I\textunderscore UNLINK} will fail with \texttt{errno} set to the value in the message.

\texttt{I\_PLINK} Connects two streams, where \texttt{fdles} is the file descriptor of the stream connected to the multiplexing driver, and \texttt{arg} is the file descriptor of the STREAM connected to another driver. The STREAM designated by \texttt{arg} gets connected via a persistent link below the multiplexing driver. \texttt{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 \texttt{fdles} 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 \texttt{I\_PUNLINK}) on success, and \texttt{-1} on failure. On failure, \texttt{errno} is set to one of the following values:

- \texttt{ENXIO} Hangup received on \texttt{fdles}.
- \texttt{ETIME} Time out before acknowledgement message was received at the STREAM head.
- \texttt{EAGAIN} Unable to allocate STREAMS storage to perform the \texttt{I\_PLINK}.
- \texttt{EBADF} \texttt{arg} is not a valid, open file descriptor.
- \texttt{EINVAL} \texttt{fdles} does not support multiplexing.
- \texttt{EINVAL} \texttt{arg} is not a STREAM or is already linked under a multiplexor.
- \texttt{EINVAL} The specified link operation would cause a “cycle” in the resulting configuration; that is, if a driver would be linked into the multiplexing configuration in more than one place.
- \texttt{EINVAL} \texttt{fdles} is the file descriptor of a pipe or FIFO.

An \texttt{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 \texttt{fdles}. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, \texttt{I\_PLINK} will fail with \texttt{errno} set to the value in the message.

\texttt{I\_PUNLINK} Disconnects the two streams specified by \texttt{fdles} and \texttt{arg} that are connected with a persistent link. \texttt{fdles} is the file descriptor of the STREAM connected to the multiplexing driver. \texttt{arg} is the multiplexor ID number that was returned by \texttt{I\_PLINK} when a STREAM was linked below the multiplexing driver. If \texttt{arg} is \texttt{MUXID\_ALL} then all streams that are persistent links to \texttt{fdles} are
disconnected. As in \_I\_PLINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, \_errno is set to one of the following values:

- **ENXIO**: Hangup received on \_fildes.
- **ETIME**: Time out before acknowledgement message was received at the STREAM head.
- **EAGAIN**: Unable to allocate buffers for the acknowledgement message.
- **EINVAL**: Invalid multiplexor ID number.
- **EINVAL**: \_fildes is the file descriptor of a pipe or FIFO.

An \_I\_PUNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request if a message indicating an error or a hangup is received at the STREAM head of \_fildes. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, \_I\_PUNLINK will fail with \_errno set to the value in the message.

**RETURN VALUES**

Unless specified otherwise above, the return value from \_ioctl() is 0 upon success and -1 upon failure, with \_errno set as indicated.

**SEE ALSO**

intro(3), close(2), fcntl(2), getmsg(2), ioctl(2), open(2), poll(2), putmsg(2), read(2), write(2), signal(3C), signal(3HEAD), 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 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...
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_BIND_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 of 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* (I/O 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 *ioctls* enable or disable the transmission of data on the stream. By default, transmission is enabled. The SL_DATA_ENABLE and SL_DATA_DISABLE *ioctls* 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>IA</td>
</tr>
</tbody>
</table>

**SEE ALSO** attributes(5), dlpi(7P), rns_smt(7D)
The `symhisl` host bus adapter driver supports the following SCSI devices: SYM22910 (channel A and B support SE and LVD mode), SYM21002 (channel A supports SE only, channel B supports both SE and LVD mode.)

The `symhisl` host bus adapter driver is a SCSA compliant nexus driver that supports the LSI Logic (formerly Symbios, Inc) SYM53C896 SCSI chipset. It supports the standard functions provided by the SCSA interface such as tagged queuing and untagged queuing, 16 bit Wide transfer, Fast/Ultra/Ultra2 synchronous transfer, and auto request sense, but it does not support linked commands.

The `symhisl` hardware (SYM53C896) supports Wide, Fast, Ultra, and Ultra2 SCSI synchronous speeds. The maximum SCSI bandwidth that SYM53C896 support is 80 MB/sec.

### Known Problems and Limitations
- For Ultra2 SCSI LVD transfer, use an Ultra2 SCSI LVD-compliant SCSI cable. For best performance, keep the devices six to eight inches apart.
- If both the 53C896 chip and an older 53C8xx chip are embedded on the motherboard, the system BIOS recognizes the older 53C8xx chip first when the system is booted. All 53C8xx chips embedded on your motherboard are then recognized by the Symbios SDMS BIOS. To prevent the older BIOS utility from starting, reflash the older 53C8xx BIOS of the add-in controller. You can download the latest Symbios SDMS BIOS from [http://www.symbios.com/techsupport/pci_sw.htm#Symbios CPI BIOS & Flash Utility](http://www.symbios.com/techsupport/pci_sw.htm#Symbios CPI BIOS & Flash Utility).
- Due to a hardware problem in revisions B0 and C0 of the 53C896 chip, the `symhisl` driver does not work properly with 64-bit PCI slots. The `symhisl` driver issues a SCSI bus reset when the problem occurs. To avoid this problem, upgrade to revision C1 of the 53C896 chip. Other revisions of the 53C896 chip might exhibit hardware incompatibilities with earlier PCI chips and designs. For a complete list of errata about each revision of the 53C896 chip, see the LSI Logic Web site at [http://www.lsi.com](http://www.lsi.com).
- Under some conditions, LVD technology may exhibit firmware incompatibilities. If you experience SCSI errors with a correctly configured LVD SCSI bus, contact the manufacturer for updated firmware.

### Configuration
Configure the `symhisl` driver by defining properties in `symhisl.conf`. These properties override the global SCSI settings. The `symhisl` driver supports the following user-modifiable properties:

- `scsi-options`
- `target<n>-scsi-options`
- `scsi-reset-delay`
- `scsi-watchdog-tick`
The symhisl(7D) command supports various options and flags for configuring the SCSI bus.

### Options

- **scsi-initiator-id**
  - The bus ID of the HBA.

- **symFlags**
  - Driver specific bit-mask that can be used to enable or disable driver properties.

  - **bit 0**
    - When set, the driver will not reset the SCSI bus at initialization. Certain CD-ROM, tape, and other devices will not work properly when this bit is set. The default state for this bit is cleared.

  - **bit 1**
    - When set, the driver will not export the DMI ioctl interface. Only set this bit if you want to disable the ioctl interface for security reasons. The default state for this bit is cleared.

  - **bit 2**
    - When set, the driver will disable 64-bit addressing capability. When clear, the driver will enable 64-bit addressing capability. The default state for this bit is cleared.

Refer to `scsi_hba_attach(9F)` for more information on driver configuration.

### Examples

**EXAMPLE 1** Edit the file `/kernel/drv/symhisl.conf` and add the following line:

```
scsi-options=0x78;
```

**EXAMPLE 2** This disables tagged queuing, Fast/Ultra/Ultra2 SCSI and wide mode for all symhisl instances.

The following example enables an option for one specific symhisl (refer to `driver.conf(4)` and `pci(4)` for more details):

```
name="symhisl" parent="/pci@1f,4000"
  unit-address="3"
  target1-scsi-options=0x58
```
symhisl(7D)

**EXAMPLE 2** This disables tagged queuing, Fast/Ultra/Ultra2 SCSI and wide mode for all symhisl instances. (Continued)

```plaintext
scsi-options=0x178 scsi-initiator-id=6;
```

**EXAMPLE 3** Note that the initiator ID can only be changed for symhisl adapters that don’t use the LSI Logic Boot ROM Configuration Utility. For adapters that can use the LSI Logic Boot ROM Configuration Utility, `scsi-initiator-id` will have no effect.

The example above sets `scsi-options` for target 1 to `0x58` and all other targets on this SCSI bus to `0x178`.

The physical 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
```

**EXAMPLE 4** 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.

`scsi-options` specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` (for all symhisl instances) per bus have the lowest precedence.

The system needs to be rebooted before the specified `scsi-options` take effect.

### Driver Capabilities

The target driver needs to set capabilities in the symhisl driver in order to enable some driver features. The target driver can query and modify these capabilities: disconnect, synchronous, wide-xfer, tagged-qing, and auto-rqsense. All other capabilities can only be queried.

By default, tagged-qing capabilities are disabled, while disconnect, synchronous, wide-xfer, auto-rqsense, and untagged-qing are enabled. These capabilities can only have binary values (0 or 1).

The target driver needs to enable tagged-qing explicitly. The untagged-qing capability is always enabled and its value cannot be modified.

Whenever there is a conflict between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

### FILES

- `/kernel/drv/symhisl` ELF Kernel Module
- `/kernel/drv/symhisl.conf` Required configuration file

### ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
symhisl(7D)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Limited to PCI-based systems with LSI Logic (formerly Symbios Inc) SYM53C896 SCSI I/O processors.</td>
</tr>
</tbody>
</table>

SEE ALSO
prtconf(1M), driver.conf(4), pci(4), attributes(5), scsi_abort(9F),
scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F),
scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S),
scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

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

LSI Logic Corporation, SYM53C896 PCI-SCSI I/O Processor

NOTES
The symhisl hardware (SYM53C896) supports Wide, Fast, Ultra, and Ultra2 SCSI synchronous speeds. The maximum SCSI bandwidth is 80 MB/sec.
sysmsg(7D)

NAME    sysmsg – system message routing to console devices

SYNOPSIS /dev/sysmsg

DESCRIPTION The file /dev/sysmsg routes output to a variable set of console devices. Writes to /dev/sysmsg are always directed to the system console /dev/console, and are in addition directed to a set of auxiliary console devices managed by consadm(1M).

Only root has permission to write to this device.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcsr</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

SEE ALSO consadm(1M), syslogd(1M), attributes(5), console(7D)
### NAME
t1394dcam – IEEE 1394 bus-based digital camera

### DESCRIPTION
t1394dcam is a digital camera which conforms to the 1394 Trade Association Camera Working Group’s 1394-based Digital Camera Specification V1.04. The video modes, framerates, and features supported by the camera are determined by the camera’s manufacturer; the camera facilitates the ability to query which of these are supported.

The driver supports camera initialization, attribute acquisition and establishment, and the ability to enable and disable the streaming of digital video into a kernel resident memory buffer via the following `ioctl(2)` commands:

- T1394_DCAM_CMD_CAM_RESET
- T1394_DCAM_CMD_PARAM_GET
- T1394_DCAM_CMD_PARAM_SET
- T1394_DCAM_CMD_FRAME_RCV_START
- T1394_DCAM_CMD_FRAME_RCV_STOP
- 1394_DCAM_CMD_RING_BUFF_FLUSH
- T1394_DCAM_CMD_FRAME_SEQ_NUM_COUNT_RESET

### FILES

- `/dev/1394/t1394dcam0`
  Device feature and video control file
- `/dev/1394/t1394dcamctl0`
  Device feature control file

### SEE ALSO

- `1394-based Digital Camera Specification V1.04`
TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol ("IP"), or the Internet Protocol Version 6 ("IPv6"), the Internet protocol family’s internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a SOCK_STREAM socket type, or using the Transport Level Interface ("TLI") where it supports the connection-oriented (T_COTS_ORD) service type.

TCP uses IP’s host-level addressing and adds its own per-host collection of “port addresses.” The endpoints of a TCP connection are identified by the combination of an IP or IPv6 address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See inet(7P) and inet6(7p) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either “active” or “passive”. Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP or IPv6 address and TCP port number bound with the bind(3SOCKET) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the listen(3SOCKET) system call after binding the socket with bind(). This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the accept(3SOCKET) system call. Active sockets use the connect(3SOCKET) call after binding to initiate connections.

By using the special value INADDR_ANY with IP, or the unspecified address (all zeroes) with IPv6, the local IP address can be left unspecified in the bind() call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP or IPv6 address will be bound at connection time to the address of the network interface used to service the connection.

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()` and tested with `getsockopt()`. The option level for the `setsockopt()` call is the protocol number for TCP, available from `getprotobyname()`. Another socket level option, `SO_RCVBUF`, can be used to control the window that TCP advertises to the peer. IP level options may also be used with TCP. See `ip(7P)` and `ip6(7P)`.

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of `send()`. The caller may mark one byte as “urgent” with the `MSG_OOB` flag to `send()`. 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()` 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()` 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:

- An application can set `SO_SNDBUF` or `SO_RCVBUF` size in the `setsockopt()` option to be larger than 64K. This must be done before the program calls `listen()` or `connect()`, because the window scale option is negotiated when the connection is established. Once the connection has been made, it is too late to increase the send or receive window beyond the default TCP limit of 64K.
For all applications, use `ndd` 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.

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 timestamp option will not be set when connecting to a remote system. The default for `tcp_tstamp_always` is 0.

- Regardless of the value of `tcp_tstamp_always`, the time stamp option will always be included in a connect acknowledgement (and all succeeding packets) if the connecting system has used the time stamp option.

Use the following procedure to turn on the time stamp option only when the window scale option is in effect:

- Use `ndd` to modify the configuration parameter `tcp_tstamp_if_wscale`. Setting `tcp_tstamp_if_wscale` to 1 will cause the time stamp option to be set when connecting to a remote system, if the window scale option has been set. If `tcp_tstamp_if_wscale` is 0, the time stamp option will not be set when connecting to a remote system. The default for `tcp_tstamp_if_wscale` is 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 HIGHLY recommend 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 /etc/init.d/inetinit script contains commands which configure initial sequence number generation. The script reads the value contained in the configuration file /etc/default/inetinit to determine which method to use.

The /etc/default/inetinit file is an unstable interface, and may change in future releases.

TCP may be configured to report some information on connections that terminate by means of an RST packet. By default, no logging is done. If the ndd(1M) parameter tcp_trace is set to 1, then trace data is collected for all new connections established after that time.

The trace data consists of the TCP headers and IP source and destination addresses of the last few packets sent in each direction before RST occurred. Those packets are logged in a series of strlog(9F) calls. This trace facility has a very low overhead, and so is superior to such utilities as snoop(1M) for non-intrusive debugging for connections terminating by means of an RST.

SEE ALSO
ndd(1M), ioctl(2), read(2), write(2), accept(3SOCKET), bind(3SOCKET),
connect(3SOCKET), getprotobyname(3SOCKET), getsockopt(3SOCKET),
listen(3SOCKET), send(3SOCKET), inet(7P), inet6(7P), ip(7P), ip6(7P)

Mathias, M. and Hahdavi, J. Pittsburgh Supercomputing Center; Ford, S. Lawrence Berkeley National Laboratory; Romanow, A. Sun Microsystems, Inc. RFC 2018, TCP Selective Acknowledgement Options, October 1996.

Bellovin, S., RFC 1948, Defending Against Sequence Number Attacks, May 1996.


DIAGNOSTICS
A socket operation may fail if:

EISCONN A connect() operation was attempted on a socket on which a connect() operation had already been performed.

ETIMEDOUT A connection was dropped due to excessive retransmissions.

ECONNRESET The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
ECONNREFUSED  The remote peer actively refused connection establishment (usually because no process is listening to the port).

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

EADDRNOTAVAIL  A bind() operation was attempted on a socket with a network address for which no network interface exists.

EACCES        A bind() operation was attempted with a “reserved” port number and the effective user ID of the process was not the privileged user.

ENOBUFs       The system ran out of memory for internal data structures.
tcx(7D)

NAME
tcx – 24-bit SBus color memory frame buffer

SYNOPSIS
SUNW,tcx@sbus-slot,offset:tcxX

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).

FBIOGATTR FBIOGCURSOR
FBIOGTYPE FBIOSCURPOS
FBIOFUTCMP FBIOGSCURPOS
FBIOGETCMAP FBIOGCURMAX
FBIOSATTR FBIOGXINFO
FBIOSVIDEO FBIOMONINFO
FBIOVVIDEO FBIOVRTOFFSET
FBIOVERTICAL VIS_GETIDENTIFIER
FBIOSCURSOR

VIS_GETIDENTIFIER returns "SUNW,tcx".

Emulation mode (FBIOGATTR, FBIOSATTR) may be either FBTYPE_SUN3COLOR or
FBTYPE_MEMCOLOR. Set emulation mode to 21 (FBTYPE_LASTPLUSONE) to turn
emulation off. Changes to emulation mode (via FBIOSATTR) 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.

FBIOFUTCMP 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.

Device and Network Interfaces 559
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)
This release supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is using function calls (the preferred interface) described in termios(3C) or ioctl commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, 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.

A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process’s process group is assigned as foreground process group.

Background process groups in the controlling process’s session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups.

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.

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.

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The MIN and TIME values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user). TIME is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case, TIME serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between MIN and TIME is as follows: as soon as one character is received, the intercharacter timer is started. If MIN characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user. Note that if TIME expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0), the read sleeps until the MIN and TIME mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN characters are received (the pending read sleeps until MIN characters are received). A program that uses this case
to read record based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since MIN = 0, TIME no longer represents an intercharacter timer: it now serves as a read timer that is activated as soon as a `read` is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within TIME *.10 seconds after the read is initiated, the read returns with zero characters.

Case D: MIN = 0, TIME = 0

In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Some points to note about MIN and TIME:

- In the following explanations, note that the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case, where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.

- Also note that in case A (MIN > 0, TIME > 0), TIME represents an intercharacter timer, whereas in case C (MIN = 0, TIME > 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, then 20 characters will be returned to the user.
Writing Characters

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue is drained down to some threshold, the program is resumed.

Special Characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

- **INTR** (Control-c or ASCII ETX) generates a SIGINT signal. SIGINT is sent to all foreground processes associated with the controlling terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed upon location. (See `signal(3HEAD)`).

- **QUIT** (Control-\| or ASCII FS) generates a SIGQUIT signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called `core`) will be created in the current working directory.

- **ERASE** (DEL) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

- **WERASE** (Control-w or ASCII ETX) erases the preceding “word”. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

- **KILL** (Control-u or ASCII NAK) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.

- **REPRINT** (Control-r or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.

- **EOF** (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (that is, the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

- **NL** (ASCII LF) is the normal line delimiter. It cannot be changed or escaped.

- **EOL** (ASCII NULL) is an additional line delimiter, like NL. It is not normally used.

- **EOL2** is another additional line delimiter.
SWITCH (Control-z or ASCII EM) is used only when sh1 layers is invoked.

SUSP (Control-z or ASCII SUB) generates a SIGTSTP signal. SIGTSTP stops all processes in the foreground process group for that terminal.

DSUSP (Control-y or ASCII EM). It generates a SIGTSTP signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.

STOP (Control-s or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.

DISCARD (Control-o or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.

LNEXT (Control-v or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (for example KILL, QUIT ). The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding backslash (‘ \ ’) character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

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.

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:

```c
  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>Relative Position</th>
<th>Subscript Name</th>
<th>Typical 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>VEOl2</td>
<td>NUL</td>
</tr>
<tr>
<td>7</td>
<td>VWSTCH</td>
<td>NUL</td>
</tr>
<tr>
<td>8</td>
<td>VSTART</td>
<td>NUL</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>
</tbody>
</table>
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 ISTRI P is set, valid input characters are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper case, alphabetic character is translated into the corresponding lower case character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If IXANY is set, any input character restarts output that has been suspended.

If IXOFF is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If IMAXBEL is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If IMAXBEL is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

Output Modes

The c_oflag field specifies the system treatment of output:

- **OPPOST**: Post-process output.
- **OLCUC**: Map lower case to upper on output.
- **ONLCK**: 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
Select carriage-return delays:

CR0
CR1
CR2
CR3

Select horizontal tab delays or tab expansion:

TAB0
TAB1
TAB2
TAB3  Expand tabs to spaces
XTABS  Expand tabs to spaces

Select backspace delays:

BS0
BS1

Select vertical tab delays:

VT0
VT1

Select form feed delays:

FF0
FF1

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If OLCUC is set, a lower case alphabetic character is transmitted as the corresponding upper case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted when at column 0 (first position). If ONRET is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.
The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise it is NULL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

### Control Modes

The `c_cflag` field describes the hardware control of the terminal:

<table>
<thead>
<tr>
<th>CBAUD</th>
<th>Baud rate:</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>
<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>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------</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>
<tr>
<td>SIZE</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>STOP</td>
<td>Send two stop bits, else one</td>
</tr>
<tr>
<td>READ</td>
<td>Enable receiver</td>
</tr>
<tr>
<td>PAREN</td>
<td>Parity enable</td>
</tr>
<tr>
<td>PARODD</td>
<td>Odd parity, else even</td>
</tr>
<tr>
<td>HUPCL</td>
<td>Hang up on last close</td>
</tr>
<tr>
<td>LOCAL</td>
<td>Local line, else dial-up</td>
</tr>
<tr>
<td>BAUD</td>
<td>Input baud rate, if different from output rate</td>
</tr>
<tr>
<td>PAREXT</td>
<td>Extended parity for mark and space parity</td>
</tr>
<tr>
<td>TSOFF</td>
<td>Enable inbound hardware flow control</td>
</tr>
<tr>
<td>TSCS</td>
<td>Enable outbound hardware flow control</td>
</tr>
<tr>
<td>BAUD</td>
<td>Bit to indicate output speed &gt; B38400</td>
</tr>
<tr>
<td>BAUD</td>
<td>Bit to indicate input speed &gt; B38400</td>
</tr>
</tbody>
</table>

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.
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 CIBAUD 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 |= CBAUDEXT;
        ispeed -= ((CIBAUD >> IBSHIFT) + 1);
    } else
        termios_p->c_cflag &= ~CBAUDEXT;
    termios_p->c_cflag = (termios_p->c_cflag & ~CBAUD) |
```
The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled, and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise, even parity is used.

If CREAD is set, the receiver is enabled. Otherwise, no characters are received.

If HUPCL is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control; otherwise, modem control is assumed.

If CRTSXOFF is set, inbound hardware flow control is enabled.

If CRTSCTS is set, outbound hardware flow control is enabled.

The four possible combinations for the state of CRTSCTS and CRTSXOFF bits and their interactions are described below.

Case A: CRTSCTS off, CRTSXOFF off. In this case the hardware flow control is disabled.

Case B: CRTSCTS on, CRTSXOFF off. In this case only outbound hardware flow control is enabled. The state of CTS signal is used to do outbound flow control. It is expected that output will be suspended if CTS is low and resumed when CTS is high.

Case C: CRTSCTS off, CRTSXOFF on. In this case only inbound hardware flow control is enabled. The state of RTS signal is used to do inbound flow control. It is expected that input will be suspended if RTS is low and resumed when RTS is high.

Case D: CRTSCTS on, CRTSXOFF on. In this case both inbound and outbound hardware flow control are enabled. Uses the state of CTS signal to do outbound flow control and RTS signal to do inbound flow control.

Local Modes

The c_lflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG
Enable signals.

ICANON
Canonical input (erase and kill processing).

XCASE
Canonical upper/lower presentation.
ECHO  Enable echo.
ECHOE  Echo erase character as BS-SP-BS &.
ECHOK  Echo NL after kill character.
ECHONL Echo NL.
NOFLSH Disable flush after interrupt or quit.
TOSTOP Send SIGTTOU for background output.
ECHOCtrl Echo control characters as char, delete as ^?.
ECHOPRT Echo erase character as character erased.
ECHOKill BS-SP-BS erase entire line on line kill.
FLUSHO Output is being flushed.
PENDIN Retype pending input at next read or input character.
IBXCTN Enable extended (implementation-defined) functions.

If \texttt{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 \texttt{ISIG} is not set, no checking is done. Thus, these special input functions are possible only if \texttt{ISIG} is set.

If \texttt{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 \texttt{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 \texttt{XCASE} is set and \texttt{ICANON} is set, an upper case letter is accepted on input if preceded by a backslash (\textbackslash) character, and is output preceded by a backslash (\textbackslash) character. In this mode, the following escape sequences are generated on output and accepted on input:

<table>
<thead>
<tr>
<th>FOR:</th>
<th>USE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>l</td>
<td>!</td>
</tr>
<tr>
<td>=~</td>
<td>^</td>
</tr>
<tr>
<td>[</td>
<td>{</td>
</tr>
<tr>
<td>]</td>
<td>}</td>
</tr>
</tbody>
</table>
For example, input A as `\a`, `\n` as `\n`, and `\N` as `\\n`.

If `ECHO` is set, characters are echoed as received.

When `ICANON` is set, the following echo functions are possible.

- If `ECHO` and `ECHOE` are set, and `ECHOPRT` is not set, the `ERASE` and `WERASE` characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.

- If `ECHO`, `ECHOPRT`, and `IEXTEN` are set, the first `ERASE` and `WERASE` character in a sequence echoes as a `'\n'` (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 `ECHOK` 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 `ECHOPR`a).

- If `ECHOK` is set, and `ECHOE` 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 `ECHOCNTL` 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).

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.

The number of lines and columns on the terminal’s display is specified in the winsize structure defined by sys/termios.h and includes the following members:

- unsigned short ws_row; /* rows, in characters */
- unsigned short ws_col; /* columns, in characters */
- unsigned short ws_xpixel; /* horizontal size, in pixels */
- unsigned short ws_ypixel; /* vertical size, in pixels */

The SunOS/SVR4 termio structure is used by some ioctl's; it is defined by sys/termio.h and includes the following members:

- unsigned short c_iflag; /* input modes */
- unsigned short c_oflag; /* output modes */
- unsigned short c_cflag; /* control modes */
- unsigned short c_lflag; /* local modes */
- char c_line; /* line discipline */
- unsigned char c_cc[NCC]; /* control chars */

The special control characters are defined by the array c_cc. The symbolic name NCC is the size of the Control-character array and is also defined by termio.h. The relative positions, subscript names, and typical default values for each function are as follows:

<table>
<thead>
<tr>
<th>Relative Positions</th>
<th>Subscript Names</th>
<th>Typical Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VINTR</td>
<td>EXT</td>
</tr>
<tr>
<td>1</td>
<td>VQUIT</td>
<td>PS</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>
</tbody>
</table>
The MIN values is stored in the VMIN element of the c_cc array; the TIME value is stored in the VTIME element of the c_cc array. The VMIN element is the same element as the VEOF element; the VTIME element is the same element as the VEOL element.

The calls that use the termio structure only affect the flags and control characters that can be stored in the termio structure; all other flags and control characters are unaffected.

Modem Lines

On special files representing serial ports, modem control lines can be read. Control lines (if the underlying hardware supports it) may also be changed. Status lines are read-only. The following modem control and status lines may be supported by a device; they are defined by sys/termios.h:

- TIOCM_LE: line enable
- TIOCM_DTR: data terminal ready
- TIOCM_RTS: request to send
- TIOCM_ST: secondary transmit
- TIOCM_SR: secondary receive
- TIOCM_CTS: clear to send
- TIOCM_CAR: carrier detect
- TIOCM_RNG: ring
- TIOCM_DSR: data set ready

TIOCM_CD is a synonym for TIOCM_CAR, and TIOCM_RI is a synonym for TIOCM_RNG. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

The software carrier mode can be enabled or disabled using the TIOCSSOFTCAR ioctl. If the software carrier flag for a line is off, the line pays attention to the hardware carrier detect (DCD) signal. The tty device associated with the line cannot be opened until DCD is asserted. If the software carrier flag is on, the line behaves as if DCD is always asserted.

The software carrier flag is usually turned on for locally connected terminals or other devices, and is off for lines with modems.
To be able to issue the `TIOCGSOFTCAR` and `TIOCSSOFTCAR` ioctl calls, the tty line should be opened with `O_NDELAY` so that the `open(2)` will not wait for the carrier.

The initial `termios` values upon driver open is configurable. This is accomplished by setting the “ttymodes” property in the file `/kernel/drv/options.conf`. Since this property is assigned during system initialization, any change to the “ttymodes” property will not take effect until the next reboot. The string value assigned to this property should be in the same format as the output of the `stty(1)` command with the `-g` option.

If this property is undefined, the following `termios` modes are in effect. The initial input control value is `BRKINT`, `ICRNL`, `IXON`, `IMAXBEL`. The initial output control value is `OPOST`, `ONLCR`, `TAB3`. The initial hardware control value is `B9600`, `CS8`, `CREAD`. The initial line-discipline control value is `ISIG`, `ICANON`, `IEXTEN`, `ECHO`, `ECHOK`, `ECHOE`, `ECHOKE`, `ECHONL`.

### Default Values

The `ioctl` supported by devices and STREAMS modules providing the `termios(3C)` interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on `termios`.

- **TCGETS**: The argument is a pointer to a `termios` structure. The current terminal parameters are fetched and stored into that structure.
- **TCSETS**: The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.
- **TCSETSW**: The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
- **TCSETSF**: The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
- **TCGETA**: The argument is a pointer to a `termio` structure. The current terminal parameters are fetched, and those parameters that can be stored in a `termio` structure are stored into that structure.
- **TCSETA**: The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change is immediate.
- **TCSETAW**: The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change occurs after all
characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

**TCSETAF**
The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

**TCSBRK**
The argument is an `int` value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).

**TCXONC**
Start/stop control. The argument is an `int` value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.

**TCFLSH**
The argument is an `int` value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

**TIOCGPGRP**
The argument is a pointer to a `pid_t`. Set the value of that `pid_t` to the process group ID of the foreground process group associated with the terminal. See `termios(3C)` for a description of `TCGETPGRP`.

**TIOCSPGRP**
The argument is a pointer to a `pid_t`. Associate the process group whose process group ID is specified by the value of that `pid_t` with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error `EPERM` is returned. See `termios(3C)` 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`.

**TIOCGWINSZ**
The argument is a pointer to a `winsize` structure. The terminal driver’s notion of the terminal size is stored into that structure.

**TIOCSDWIN**
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.

**TIOCMIBS**
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.
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.

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.

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.

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.

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.

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.

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.

The argument is a pointer to a struct ppsclockev. This structure contains the following members:

```c
struct timeval tv;
uint32_t serial; "tv" is the system clock timestamp when the event (pulse on the DCD pin) occurred. "serial" is the ordinal of the event, which each consecutive event being assigned the next ordinal. The first event registered gets a "serial" value of 1. The TIOCGPPSEV returns the last event registered; multiple calls will persistently return the same event until a new one is registered. In addition to time stamping and saving the event, if it is of one-second period and of consistently high accuracy, the local system clock will automatically calibrate to it.
```

Files in or under /dev

SEE ALSO

stty(1), fork(2), getsid(2), ioctl(2), setsid(2), sigaction(2), signal(3C), termios(3C), signal(3HEAD), streamio(7I)

Device and Network Interfaces 581
termiox(7I)

NAME
termiox – extended general terminal interface

DESCRIPTION
The extended general terminal interface supplements the termio(7I) general terminal interface by adding support for asynchronous hardware flow control, isochronous flow control and clock modes, and local implementations of additional asynchronous features. Some systems may not support all of these capabilities because of either hardware or software limitations. Other systems may not permit certain functions to be disabled. In these cases the appropriate bits will be ignored. See <sys/termiox.h> for your system to find out which capabilities are supported.

Hardware Flow Control Modes
Hardware flow control supplements the termio(7I) IXON, IXOFF, and IXANY character flow control. Character flow control occurs when one device controls the data transfer of another device by the insertion of control characters in the data stream between devices. Hardware flow control occurs when one device controls the data transfer of another device using electrical control signals on wires (circuits) of the asynchronous interface. Isochronous hardware flow control occurs when one device controls the data transfer of another device by asserting or removing the transmit clock signals of that device. Character flow control and hardware flow control may be simultaneously set.

In asynchronous, full duplex applications, the use of the Electronic Industries Association's EIA-232-D Request To Send (RTS) and Clear To Send (CTS) circuits is the preferred method of hardware flow control. An interface to other hardware flow control methods is included to provide a standard interface to these existing methods.

The EIA-232-D standard specified only unidirectional hardware flow control - the Data Circuit-terminating Equipment or Data Communications Equipment (DCE) indicates to the Data Terminal Equipment (DTE) to stop transmitting data. The termiox interface allows both unidirectional and bidirectional hardware flow control; when bidirectional flow control is enabled, either the DCE or DTE can indicate to each other to stop transmitting data across the interface. Note: It is assumed that the asynchronous port is configured as a DTE. If the connected device is also a DTE and not a DCE, then DTE to DTE (for example, terminal or printer connected to computer) hardware flow control is possible by using a null modem to interconnect the appropriate data and control circuits.

Clock Modes
Isochronous communication is a variation of asynchronous communication whereby two communicating devices may provide transmit and/or receive clock signals to one another. Incoming clock signals can be taken from the baud rate generator on the local isochronous port controller, from CCITT V.24 circuit 114, Transmitter Signal Element Timing - DCE source (EIA-232-D pin 15), or from CCITT V.24 circuit 115, Receiver Signal Element Timing - DCE source (EIA-232-D pin 17). Outgoing clock signals can be sent on CCITT V.24 circuit 113, Transmitter Signal Element Timing - DTE source (EIA-232-D pin 24), on CCITT V.24 circuit 128, Receiver Signal Element Timing - DTE source (no EIA-232-D pin), or not sent at all.
In terms of clock modes, traditional asynchronous communication is implemented simply by using the local baud rate generator as the incoming transmit and receive clock source and not outputting any clock signals.

The parameters that control the behavior of devices providing the termio interface are specified by the termio structure defined in the <sys/termio.h> header. Several ioctl(2) system calls that fetch or change these parameters use this structure:

```c
#define NFF 5
struct termio {  
  unsigned short x_hflag; /* hardware flow control modes */  
  unsigned short x_cflag; /* clock modes */  
  unsigned short x_rflag[NFF]; /* reserved modes */  
  unsigned short x_sflag; /* spare local modes */  
};
```

The x_hflag field describes hardware flow control modes:

- RTSXOFF 0000001 Enable RTS hardware flow control on input.
- CTSXON 0000002 Enable CTS hardware flow control on output.
- DTRXOFF 0000004 Enable DTR hardware flow control on input.
- CDXON 0000010 Enable CD hardware flow control on output.
- ISXOFF 0000020 Enable isochronous hardware flow control on input.

The EIA-232-D DTR and CD circuits are used to establish a connection between two systems. The RTS circuit is also used to establish a connection with a modem. Thus, both DTR and RTS are activated when an asynchronous port is opened. If DTR is used for hardware flow control, then RTS must be used for connectivity. If CD is used for hardware flow control, then CTS must be used for connectivity. Thus, RTS and DTR (or CTS and CD) cannot both be used for hardware flow control at the same time. Other mutual exclusions may apply, such as the simultaneous setting of the termio(7I) HUPCL and the termio DTRXOFF bits, which use the DTE ready line for different functions.

Variations of different hardware flow control methods may be selected by setting the appropriate bits. For example, bidirectional RTS/CTS flow control is selected by setting both the RTSXOFF and CTSXON bits and bidirectional DTR/CTS flow control is selected by setting both the DTRXOFF and CTSXON. Modem control or unidirectional CTS hardware flow control is selected by setting only the CTSXON bit.

As previously mentioned, it is assumed that the local asynchronous port (for example, computer) is configured as a DTE. If the connected device (for example, printer) is also a DTE, it is assumed that the device is connected to the computer’s asynchronous port using a null modem that swaps control circuits (typically RTS and CTS). The connected DTE drives RTS and the null modem swaps RTS and CTS so that the
remote RTS is received as CTS by the local DTE. In the case that CTSXON is set for hardware flow control, printer’s lowering of its RTS would cause CTS seen by the computer to be lowered. Output to the printer is suspended until the printer’s raising of its RTS, which would cause CTS seen by the computer to be raised.

If RTSXOFF is set, the Request To Send (RTS) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the Request To Send (RTS) line. If the RTS line is lowered, it is assumed that the connected device will stop its output until RTS is raised.

If CTSXON is set, output will occur only if the Clear To Send (CTS) circuit (line) is raised by the connected device. If the CTS line is lowered by the connected device, output is suspended until CTS is raised.

If DTRXOFF is set, the DTE Ready (DTR) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the DTE Ready (DTR) line. If the DTR line is lowered, it is assumed that the connected device will stop its output until DTR is raised.

If CDXON is set, output will occur only if the Received Line Signal Detector (CD) circuit (line) is raised by the connected device. If the CD line is lowered by the connected device, output is suspended until CD is raised.

If ISXOFF is set, and if the isochronous port needs to have its input stopped, it will stop the outgoing clock signal. It is assumed that the connected device is using this clock signal to create its output. Transit and receive clock sources are programmed using the x_cflag fields. If the port is not programmed for external clock generation, ISXOFF is ignored. Output isochronous flow control is supported by appropriate clock source programming using the x_cflag field and enabled at the remote connected device.

The x_cflag field specifies the system treatment of clock modes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMTCLK</td>
<td>000007</td>
<td>Transmit clock source:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get transmit clock from internal baud rate generator.</td>
</tr>
<tr>
<td>XCI BRG</td>
<td>000000</td>
<td>Get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.</td>
</tr>
<tr>
<td>XCTSET</td>
<td>000001</td>
<td>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>XCRSET</td>
<td>000002</td>
<td>Get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.</td>
</tr>
<tr>
<td>RCV CLK</td>
<td>000070</td>
<td>Receive clock source:</td>
</tr>
<tr>
<td>RCIB RG</td>
<td>000000</td>
<td>Get receive clock from internal baud rate generator.</td>
</tr>
</tbody>
</table>
### termiox(7I)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCTSET</td>
<td>000010</td>
<td>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>000020</td>
<td>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</td>
<td>Transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source:</td>
</tr>
<tr>
<td>TSETCOFF</td>
<td>0000000</td>
<td>TSET clock not provided.</td>
</tr>
<tr>
<td>TSETCRBRG</td>
<td>000100</td>
<td>Output receive baud rate generator on circuit 113.</td>
</tr>
<tr>
<td>TSETCTBRG</td>
<td>000200</td>
<td>Output transmit baud rate generator on circuit 113.</td>
</tr>
<tr>
<td>TSETCTSET</td>
<td>000300</td>
<td>Output transmitter signal element timing (DCE source) on circuit 113.</td>
</tr>
<tr>
<td>TSETCRSET</td>
<td>000400</td>
<td>Output receiver signal element timing (DCE source) on circuit 113.</td>
</tr>
<tr>
<td>RSETCLK</td>
<td>0007000</td>
<td>Receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source:</td>
</tr>
<tr>
<td>RSETCOFF</td>
<td>0000000</td>
<td>RSET clock not provided.</td>
</tr>
<tr>
<td>RSETCRBRG</td>
<td>0001000</td>
<td>Output receive baud rate generator on circuit 128.</td>
</tr>
<tr>
<td>RSETCTBRG</td>
<td>0002000</td>
<td>Output transmit baud rate generator on circuit 128.</td>
</tr>
<tr>
<td>RSETCTSET</td>
<td>0003000</td>
<td>Output transmitter signal element timing (DCE source) on circuit 128.</td>
</tr>
<tr>
<td>RSETCRSET</td>
<td>0004000</td>
<td>Output receiver signal element timing (DCE source) on circuit 128.</td>
</tr>
</tbody>
</table>

If the XMTCLK field has a value of XCI BRG 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 RCI BRG the receive clock is taken from the hardware Internal Baud Rate Generator, as in normal asynchronous transmission. If RCVCLK = RCTSET the receive clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If RCVCLK = RCRSET the receive clock is taken from the Receiver Signal Element Timing (DCE source) circuit.
If the TSETCLK field has a value of TSETCOFF the Transmitter Signal Element Timing (DTE source) circuit is not driven. If TSETCLK = 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 = 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 Transmit Baud Rate Generator. If RSETCLK = RSETCRBRG 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.

**IOCTLs**

The ioctl(2) system calls have the form:

`ioctl (fd, command, arg) struct termiox * arg;`  The commands using this form are:

- **TCGETX**
  The argument is a pointer to a termiox structure. The current terminal parameters are fetched and stored into that structure.

- **TCSETX**
  The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

- **TCSETXW**
  The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

- **TCSETXF**
  The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

**FILES**

/dev/*
SEE ALSO  stty(1), ioctl(2), termio(7I)

NOTES  The termio(7I) system call is provided for compatibility with previous releases and its use is discouraged. Instead, the termio(7I) system call is recommended. See termio(7I) for usage information.
### NAME
 Ticlts, ticots, ticotsord – loopback transport providers

### SYNOPSIS

```
#include <sys/ticlts.h>
#include <sys/ticots.h>
#include <sys/ticotsord.h>
```

### DESCRIPTION

The devices known as ticlts, ticots, and ticotsord are “loopback transport
providers,” that is, stand-alone networks at the transport level. Loopback transport
providers are transport providers in every sense except one: only one host (the local
machine) is “connected to” a loopback network. Loopback transports present a TPI
(STREAMS-level) interface to application processes and are intended to be accessed
via the TLI (application-level) interface. They are implemented as clone devices and
support address spaces consisting of “flex-addresses,” that is, arbitrary sequences of
octets of length > 0, represented by a netbuf structure.

ticlts is a datagram-mode transport provider. It offers (connectionless) service of
type T_CLTS. Its default address size is TCL_DEFAULTADDRSZ. ticlts prints the
following error messages (see t_rcvuderr(3NSL)):

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCL_BADADDR</td>
<td>bad address specification</td>
</tr>
<tr>
<td>TCL_BADOPT</td>
<td>bad option specification</td>
</tr>
<tr>
<td>TCL_NOPEER</td>
<td>bound</td>
</tr>
<tr>
<td>TCL_PEERBADSTATE</td>
<td>peer in wrong state</td>
</tr>
</tbody>
</table>

ticots is a virtual circuit-mode transport provider. It offers (connection-oriented)
service of type T_COTS. Its default address size is TCO_DEFAULTADDRSZ. ticots
prints the following disconnect messages (see t_rcvdis(3NSL)):

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCO_NOPEER</td>
<td>no listener on destination address</td>
</tr>
<tr>
<td>TCO_PEERNOROOMONQ</td>
<td>peer has no room on connect queue</td>
</tr>
<tr>
<td>TCO_PEERBADSTATE</td>
<td>peer in wrong state</td>
</tr>
<tr>
<td>TCO_PEERINITIATED</td>
<td>peer-initiated disconnect</td>
</tr>
<tr>
<td>TCO_PROVIDERINITIATED</td>
<td>provider-initiated disconnect</td>
</tr>
</tbody>
</table>

ticotsord is a virtual circuit-mode transport provider, offering service of type
T_COTS_ORD (connection-oriented service with orderly release). Its default address
size is TCOO_DEFAULTADDRSZ. ticotsord prints the following disconnect messages
(see t_rcvdis(3NSL)):

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCOO_NOPEER</td>
<td>no listener on destination address</td>
</tr>
<tr>
<td>TCOO_PEERNOROOMONQ</td>
<td>peer has no room on connect queue</td>
</tr>
<tr>
<td>TCOO_PEERBADSTATE</td>
<td>peer in wrong state</td>
</tr>
<tr>
<td>TCOO_PEERINITIATED</td>
<td>provider-initiated disconnect</td>
</tr>
</tbody>
</table>
TCOO_PROVIDERINITIATED peer-initiated disconnect

Loopback transports support a local IPC mechanism through the TLI interface. Applications implemented in a transport provider-independent manner on a client-server model using this IPC are transparently transportable to networked environments.

Transport provider-independent applications must not include the headers listed in the synopsis section above. In particular, the options are (like all transport provider options) provider dependent.

ticlts and ticots support the same service types (T_CLTS and T_COTS) supported by the OSI transport-level model.

ticotsord supports the same service type (T_COTSORD) supported by the TCP/IP model.

FILES
/dev/ticlts
/dev/ticots
/dev/ticotsord

SEE ALSO t_rcvdis(3NSL), t_rcvuderr(3NSL)
timod(7M)

NAME    | timod – Transport Interface cooperating STREAMS module
SYNOPSIS| 
DESCRIPTION| 
# include <sys/stropts.h>

#include <sys/stropts.h>

ioctl(fildes, I_STR, &my_strioctl);

The timod module must be pushed onto only a stream terminated by a transport protocol provider that supports the TI.

All STREAMS messages, with the exception of the message types generated from the ioctl commands described below, will be transparently passed to the neighboring module or driver. The messages generated from the following ioctl commands are recognized and processed by the timod module. The format of the ioctl call is:

#include <sys/stropts.h>

- ...
- struct strioctl my_strioctl;
- ...
- strioctl.ic_cmd = cmd;
- strioctl.ic_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(3), ioctl(2), errno(3C), tirdwr(7M)

STREAMS Programming Guide

Network Interface Guide

DIAGNOSTICS

If the ioctl returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in <sys/tiuser.h>. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in <sys/errno.h> (see intro(3)).
tirdwr(7M)

NAME
tirdwr – Transport Interface read/write interface STREAMS module

SYNOPSIS
int ioctl(fd, I_PUSH, "tirdwr");

DESCRIPTION
tirdwr is a STREAMS module that provides an alternate interface to a transport
provider which supports the Transport Interface ("TI") functions of the Network
Services library (see Section 3N). This alternate interface allows a user to communicate
with the transport protocol provider using the read(2) and write(2) system calls. The
putmsg(2) and getmsg(2) system calls may also be used. However, putmsg and
getmsg can only transfer data messages between user and stream; control portions
are disallowed.

The tirdwr module must only be pushed (see I_PUSH in streamio(7I)) onto a
stream terminated by a transport protocol provider which supports the TI. After the
tirdwr module has been pushed onto a stream, none of the TI functions can be used.
Subsequent calls to TI functions cause an error on the stream. Once the error is
detected, subsequent system calls on the stream return an error with errno set to
EPROTO.

The following are the actions taken by the tirdwr module when pushed on the
stream, popped (see I_POP in streamio(7I)) off the stream, or when data passes
through it.

push
When the module is pushed onto a stream, it checks any existing
data destined for the user to ensure that only regular data
messages are present. It ignores any messages on the stream that
relate to process management, such as messages that generate
signals to the user processes associated with the stream. If any
other messages are present, the I_PUSH will return an error with
errno set to EPROTO.

write
The module takes the following actions on data that originated
from a write system call:

- All messages with the exception of messages that contain
  control portions (see the putmsg and getmsg system calls) are
  transparently passed onto the module’s downstream neighbor.
- Any zero length data messages are freed by the module and
  they will not be passed onto the module’s downstream
  neighbor.
- Any messages with control portions generate an error, and any
  further system calls associated with the stream fails with
  errno set to EPROTO.

read
The module takes the following actions on data that originated
from the transport protocol provider.

All messages with the exception of those that contain control
portions (see the putmsg and getmsg system calls) are
transparency passed onto the module’s upstream neighbor. The action taken on messages with control portions will be as follows:

- Any data messages with control portions have the control portions removed from the message before passing the message on to the upstream neighbor.
- Messages that represent an orderly release indication from the transport provider generate a zero length data message, indicating the end of file, which will be sent to the reader of the stream. The orderly release message itself is freed by the module.
- Messages that represent an abortive disconnect indication from the transport provider cause all further `write` and `putmsg` system calls to fail with `errno` set to `ENXIO`. All further `read` and `getmsg` system calls return zero length data (indicating end of file) once all previous data has been read.
- With the exception of the above rules, all other messages with control portions generate an error and all further system calls associated with the stream will fail with `errno` set to `EPROTO`. Any zero length data messages are freed by the module and they are not passed onto the module’s upstream neighbor.

**pop**

When the module is popped off the stream or the stream is closed, the module takes the following action:

- If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

**SEE ALSO**

`intro(3)`, `getmsg(2)`, `putmsg(2)`, `read(2)`, `write(2)`, `intro(3)`, `streamio(7I)`, `timod(7M)`

*STREAMS Programming Guide*

*Network Interface Guide*
tmpfs is a memory based file system which uses kernel resources relating to the VM system and page cache as a file system. Once mounted, a tmpfs file system provides standard file operations and semantics. tmpfs is so named because files and directories are not preserved across reboot or unmounts, all files residing on a tmpfs file system that is unmounted will be lost.

tmpfs file systems can be mounted with the command:

```bash
mount -F tmpfs swap directory
```

Alternatively, to mount a tmpfs file system on /tmp at multi-user startup time (maximizing possible performance improvements), add the following line to /etc/vfstab:

```bash
swap -/tmp tmpfs - yes -
```

tmpfs is designed as a performance enhancement which is achieved by caching the writes to files residing on a tmpfs file system. Performance improvements are most noticeable when a large number of short lived files are written and accessed on a tmpfs file system. Large compilations with tmpfs mounted on /tmp are a good example of this.

Users of tmpfs should be aware of some constraints involved in mounting a tmpfs file system. The resources used by tmpfs are the same as those used when commands are executed (for example, swap space allocation). This means that large sized tmpfs files can affect the amount of space left over for programs to execute. Likewise, programs requiring large amounts of memory use up the space available to tmpfs. Users running into this constraint (for example, running out of space on tmpfs) can allocate more swap space by using the `swap(1M)` command.

Another constraint is that the number of files available in a tmpfs file system is calculated based on the physical memory of the machine and not the size of the swap device/partition. If you have too many files, tmpfs will print a warning message and you will be unable to create new files. You cannot increase this limit by adding swap space.

Normal file system writes are scheduled to be written to a permanent storage medium along with all control information associated with the file (for example, modification time, file permissions). tmpfs control information resides only in memory and never needs to be written to permanent storage. File data remains in core until memory demands are sufficient to cause pages associated with tmpfs to be reused at which time they are copied out to swap.
An additional mount option can be specified to control the size of an individual tmpfs file system.

SEE ALSO

df(1M), mount(1M), mount_tmpfs(1M), swap(1M), mmap(2), mount(2), umount(2), vfstab(4)

System Administration Guide, Volume 1

DIAGNOSTICS

If tmpfs runs out of space, one of the following messages will display in the console.

directory: File system full, swap space limit exceeded
This message appears because a page could not be allocated while writing to a file. This can occur if tmpfs is attempting to write more than it is allowed, or if currently executing programs are using a lot of memory. To make more space available, remove unnecessary files, exit from some programs, or allocate more swap space using swap(1M).

directory: File system full, memory allocation failed
tmpfs ran out of physical memory while attempting to create a new file or directory. Remove unnecessary files or directories or install more physical memory.

WARNINGS

Files and directories on a tmpfs file system are not preserved across reboots or unmounts. Command scripts or programs which count on this will not work as expected.

NOTES

Compilers do not necessarily use /tmp to write intermediate files therefore missing some significant performance benefits. This can be remedied by setting the environment variable TMPDIR to /tmp. Compilers use the value in this environment variable as the name of the directory to store intermediate files.

swap to a tmpfs file is not supported.

df(1M) output is of limited accuracy since a tmpfs file system size is not static and the space available to tmpfs is dependent on the swap space demands of the entire system.
tpf provides the platform dependent functions for Solaris IA MP support. These functions adhere to the PSM Specifications. (Platform Specific Module Interface Specifications.) Tricord Systems Enterprise Servers are Intel APIC based MP platforms which run from 1 to 12 Intel processors. The tpf psm supports dynamic interrupt distribution across all processors in an MP configuration.

The psm is automatically invoked on an ESxxxx platform at system boot time.

FILES
/kernel/mach/tpf MP module.

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

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

SEE ALSO
attributes(5)
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>
ioctl(fd, I_PUSH, "ttcompat");
```

DESCRIPTION

`ttcompat` is a STREAMS module that translates the `ioctl` calls supported by the older Version 7, 4BSD, and XENIX terminal drivers into the `ioctl` calls supported by the `termio` interface (see `termio(7I)`). All other messages pass through this module unchanged; the behavior of `read` and `write` calls is unchanged, as is the behavior of `ioctl` calls other than the ones supported by `ttcompat`.

This module can be automatically pushed onto a stream using the `autopush` mechanism when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the `termios` interface be supported by the modules and the application can push the driver downstream. The `TCGETS`, `TCSETS`, and `TCSETSF` `ioctl` calls must be supported. If any information set or fetched by those `ioctl` calls is not supported by the modules and driver downstream, some of the V7/4BSD/XENIX functions may not be supported. For example, if the `CBAUD` bits in the `c_cflag` field are not supported, the functions provided by the `sg_ispeed` and `sg_ospeed` fields of the `sgttyb` structure (see below) will not be supported. If the `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 VEROSE 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</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_iflag field, are cleared in the termios structure) and input
processing (all flags in the c_iflag field other than the IXOFF and IXANY flags are cleared in the termios structure). 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_oflag 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:

- **sg_flags**: flags in termios structure
- **CBREAK**: complement of ICANON in c_lflag field
- **ECHO**: ECHO in c_lflag field
- **TANDEM**: IXOFF in c_iflag field

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by `<sys/ttold.h>`:

```c
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
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:

- **t_stopc** (VSTOP)
- **t_eofc** (VEOF)
- **t_brkc** (VEOL)

Local flags: flags in termios structure
- **LCRTBS** (not supported)
- **LPRTERA** (ECHOPRT in the c_lflag field)
- **LCRTERA** (ECHOE in the c_lflag field)
- **LTILODE** (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)
- **LPENDIN** (PENDIN in the c_lflag field)
- **LDECCCTQ** (complement of IXANY in the c_iflag field)
- **LNOFLSH** (NOFLSH 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)
ttcompat(7M)

### 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.

- **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.

- **TIOCPCH**

- **TIOSW**

- **TIOCSW**

- **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.

- **TIOCFLUSH**: The argument is a pointer to an int variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the int is treated as the logical OR of the FREAD and FWRITE flags defined by <sys/file.h>. If the FREAD bit is set, all characters waiting in input queues are flushed, and if the FWRITE bit is set, all characters waiting in output queues are flushed.

- **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.

- **TIOCSDDTR**: The argument is ignored. The Data Terminal Ready bit is set for the device.

- **TIOCCDTR**: The argument is ignored. The Data Terminal Ready bit is cleared for the device.
TIOCSTOP  The argument is ignored. Output is stopped as if the STOP character had been typed.

TIOCSTART  The argument is ignored. Output is restarted as if the START character had been typed.

TIOCGETC  The argument is a pointer to a tchars structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.

TIOCSETC  The argument is a pointer to a tchars structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

TIOCLGET  The argument is a pointer to an int. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the int pointed to by the argument.

TIOCLBIS  The argument is a pointer to an int whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLBIC  The argument is a pointer to an int whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLSET  The argument is a pointer to an int containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word.

TIOCGLTC  The argument is a pointer to an ltchars structure. The values of the appropriate characters in the terminal state are stored in that structure.

TIOCSLTC  The argument is a pointer to an ltchars structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

FIORDCHK  Returns the number of immediately readable characters. The argument is ignored.

FIONREAD  Returns the number of immediately readable characters in the int pointed to by the argument.

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 ioctls are returned as successful for the sake of compatibility. However, nothing significant is done (that is, the state of the terminal is not changed in any way).

- TIOCSETD: LDOPEN
- TIOCGETD: LDCLOSE
- DIOCSETP: LDCHG
- DIOCSETP: LDSETT
- DIOGETP: LDGETT

SEE ALSO: ioctl(2), termios(3C), ldterm(7M), termio(7I)

NOTES: TIOCBRK and TIOCCEBRK should be handled by the driver. FIONREAD and FIORDCHK are handled in the stream head.
tty(7D)

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)
tun, TUN – tunneling STREAMS module

DESCRIPTION

tun and atun are STREAMS modules that implement an IP-in-IP tunneling mechanism. IPv6-in-IPv4 and IPv4-in-IPv4 tunnels are supported.

Tunnels are configured as point-to-point interfaces. IPv4-in-IPv4 allows IPv4 packets to be encapsulated within IPv4 packets. IPv6-in-IPv4 tunnels allow IPv6 packets to be encapsulated within IPv4 packets. Both the tunnel source and the tunnel destination are required to configure these type of tunnels. Configured tunnels support encapsulated multicast packets. See ifconfig(1M) for examples of these tunnel configurations.

The atun module is used to configure automatic tunnels. It supports IPv6 packets encapsulated within IPv4 packets. An IPv4 address is required for the tunnel source of these interfaces and the IPv4 compatible IPv6 source address must match this address. IPv6 packets using this interface must have IPv4 compatible source and destination addresses. Automatic tunnels are not point-to-point, and they do not allow multicast packets to be sent. If the destination of an automatic tunnel is a router, the packets will not be forwarded.

- Network startup scripts look at /etc/hostname.ip.* to find the available tunneling interfaces.
- The same tunnel source address (tsrc) and destination address (tdst) is be used for all instances (luns) of a specific interface.
- Tunnels do not support snooping. Instead, a filter made up of the combination of addresses can be used on the physical interface to capture relevant packets.
- If there is a tunnel set up between two multicast routers, then multicast routing should be configured to use the tunnel, rather than a special multicast routing virtual interface.

The tunnel module is architected to be plumbed between two instances of IP.

The following ioctl() calls may be used to configure a tunneling interface. The ioctl() s are defined in <sys/sockio.h>. This structure is defined in <net/if.h>.

/* currently tunnels only support IPv4 or IPv6 */
enum ifta_proto {
    IFTAP_INVALID,
    IFTAP_IPV4,
    IFTAP_IPV6
};

#define IPTUN_SECINFOLEN 8
#define IPTUN_VERSION 1

/* tunnel configuration structure */
struct iftun_req {
    char ifta_lifr_name[LIFNAMSIZ]; /* if name */
    struct sockaddr_storage ifta_saddr; /* source address */
    struct sockaddr_storage ifta_daddr; /* destination address */
    uint_t ifta_flags; /* See below */
        /* IP version information is read only */
    enum ifta_proto ifta_upper; /* IP version above tunnel */
    enum ifta_proto ifta_lower; /* IP version below tunnel */
    uint_t ifta_vers; /* Version number */
    uint32_t ifta_secinfo[IFTUN_SECINFOLEN]; /* Security prefs. */
};
    /* These flags are set to indicate which members are valid */
#define IFTUN_SRC 0x01
#define IFTUN_DST 0x02
#define IFTUN_SECURITY 0x04

The ifta_vers field indicates what IPsec request structure is overlayed on top of ifta_secinfo. The current value of IFTUN_VERSION implies an overlay of ipsec_req_t. See ipsec(7P).

SIOCSTUNPARAM Set tunnel parameters. This ioctl() allows the tunnel’s source or destination address to be set. The IFTUN_SRC bit set in ta_flags indicates that the tunnel should bound to the source address supplied in ta_saddr. The source must be a valid configured interface IP address. The IFTUN_DST bit set in ta_flags indicates that the tunnel should bound to the destination address supplied in ta_daddr. The destination address must be reachable.

SIOCGTUNPARAM Get tunnel parameters. Valid fields are indicated by the returned value of ta_flags bitmask. The version of IP plumbed above or below the tunnel may be determined by inspecting ta_upper and ta_lower by comparing the members against the mutually exclusive defined values IFTAP_INVALID, IFTAP_IPV4, and IFTAP_IPV6. Currently, only IFTAP_IPV4 is supported, as IP is currently version 4.

Tunnels and DLPI

The tunnel module is a DLPI style 2 service provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPIprimitives. 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 indicates the corresponding device instance (unit) number. The device is initialized on first attach and deinitialized (stopped) on last detach.

The values returned by the module in the DL_INFO_ACK primitive in response to the DL_INFO_REQ from the user are as follows:

- The maximum SDU is usually 4196 ("ip_max_mtu - size of IP header").
The minimum SDU is 1.

The dlsap address length is 0 for configured tunnels and non-zero for automatic tunnels.

The MAC type is DL_OTHER.

The sap length value is 0.

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 0

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 tunneling module interprets the sap field within the DL_BIND_REQ as an IP "type" therefore the valid value for the sap field is IP_DL_SAP.

Once in the DL_BOUND state, the user may transmit packets through the tunnel by sending DL_UNITDATA_REQ messages to the tunnel module. Configured tunnels will encapsulate the packet with the appropriate IP header using the source and destination specified by tsrc and tdst parameters of ifconfig(1M). The tunnel module will decapsulate received packets and route them to the first open and bound stream having a sap, tsrc and tdst which matches the configured information. Packets are routed to exactly one open stream and not duplicated.

The module does not support additional primitives. DL_ERROR_ACK with the dl_error set to DL_UNSUPPORTED will be returned in the case that an unsupported DLPI primitive is encountered.

A tunnel creates what appears to be a physical interface to IP. It can be "trusted" as a physical link only so far as the underlying security protocols, if used, can be trusted. If the security associations (see ipsec(7P)) are securely set up then the tunnel can be trusted in that packets that come off the tunnel came from the peer specified in the tunnel destination. If this trust exists, per-interface IP forwarding can be used to create a Virtual Private Network ("VPN"). See ip(7P).

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>SUNWcsr (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcsr (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>
tun(7M)

SEE ALSO ifconfig(1M), attributes(5), ip(7P), ipsec(7P)

System Administration Guide, Volume 3

The uata host bus adapter driver is a nexus driver that supports the ide interface on SPARC platforms. The driver supports ultra DMA mode-4 (ATA66). The driver also attempts to set the disk and ATAPI CD-ROM drive to maximum supported speed for the device.

Currently, the uata driver supports IDE controllers CMD646U and Acer Southbridge M5229.

The uata driver supports two channels concurrently with two devices connected per channel. The devices are logically numbered from 0 to 3:

0  Master disk on primary channel
1  Slave disk on primary channel
2  Master disk on secondary channel
3  Slave disk on secondary channel

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

`ddi_get_iblock_cookie failed`
  The driver could not obtain the interrupt cookie; the attach may fail.

`Drive not ready before set_features`
  Indicates a fatal problem; the drives are not ready to be programmed and features cannot be set. (During the driver initialization process, the driver must set the features for the drive, including dma and pio.)

`Interrupt not seen after set_features`
  Indicates a fatal problem with the drive; features could not be set.

`ata_controller - set features failed`
  Indicates a fatal problem with the drive; features could not be set.

`? target %d lun %d`
  Displayed at boot up time to indicate that target <#number> was identified.
resid = %x
   A residual in data transfer and the I/O operation could not be finished completely.

ghd_timer_newstate: HBA reset failed
   Generally indicates a fatal condition; I/O operation could not be completed
   following reset of the channel.

timeout: <msgp> target = %d lun=0
   msgp could be early abort, early timeout, abort request, abort device, reset target,
   reset bus.

ata_controller - Drive not ready before command %x
   The drive did not respond before issuing the command %x to the controller;
   command %x will not be issued to the drive. (%x is the opcode for the sleep or
   standby commands, which are issued when the drive transitions between power
   management states.)

ata_controller - Command %x failed
   The command %x failed on the drive. (%x is the opcode for the sleep or standby
   commands, which are issued when the drive transitions between power
   management states.)

ata_controller - Command %x returned error
   The command %x returned error. (%x is the opcode for the sleep or standby
   commands, which are issued when the drive transitions between power
   management states.)

ata_controller - Cannot take drive %d to sleep
   The disk will not transition to sleep state. (Indicates that the driver could not set the
   device to sleep mode while performing power management functions.)

ata_controller - Cannot reset secondary/primary channel
   The disk will not transition from sleep to active state.

These messages are informational and indicate that a timeout occured for a I/O
request. The uata driver recovers from these states automatically unless there is a
fatal error.
The udfs file system is a file system type that allows user access to files on Universal Disk Format (UDF) disks from within the Solaris operating environment. Once mounted, a udfs file system provides standard Solaris file system operations and semantics. That is, users can read files, write files, and list files in a directory on a UDF device and applications can use standard UNIX system calls on these files and directories.

Because udfs is a platform-independent file system, the same media can be written to and read from by any operating system or vendor.

**Mounting File Systems**

udfs file systems are mounted using:

```
mount -F udfs -o rw/ro device-special
```

Use:

```
mount /udfs
```

if the /udfs and device special file /dev/dsk/c0t6d0s0 are valid and the following line (or similar line) appears in your /etc/vfstab file:

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

The udfs file system provides read-only support for ROM, RAM, and sequentially-recordable media and read-write support on RAM media.

The udfs file system also supports regular files, directories, and symbolic links, as well as device nodes such as block, character, FIFO, and Socket.

**SEE ALSO**

mount(1M), mount_udfs(1M), vfstab(4)

**NOTES**

Invalid characters such as "NULL" and "/" and invalid file names such as "." and ".." will be translated according to the following rule:

Replace the invalid character with an "_", then append the file name with # followed by a 4 digit hex representation of the 16-bit CRC of the original FileIdentifier. For example, the file name "..." will become "__#4C05"
NAME | udp, UDP – Internet User Datagram Protocol
SYNOPSIS | 
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_DGRAM, 0);
s = socket(AF_INET6, SOCK_DGRAM, 0);
t = t_open("/dev/udp", O_RDWR);
t = t_open("/dev/udp6", O_RDWR);

DESCRIPTION | UDP is a simple datagram protocol which is layered directly above the Internet Protocol (“IP”) or the Internet Protocol Version 6 (“IPv6”). Programs may access UDP using the socket interface, where it supports the SOCK_DGRAM socket type, or using the Transport Level Interface (“TLI”), where it supports the connectionless (T_CLTS) service type.

Within the socket interface, UDP is normally used with the sendto(), sendmsg(), recvfrom(), and recvmsg() calls (see send(3SOCKET) and recv(3SOCKET)). If the connect(3SOCKET) call is used to fix the destination for future packets, then the recv(3SOCKET) or read(2) and send(3SOCKET) or write(2) calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (“TCP”). Like TCP, UDP uses a port number along with an IP or IPv6 address to identify the endpoint of communication. The UDP port number space is separate from the TCP port number space, that is, a UDP port may not be “connected” to a TCP port. The bind(3SOCKET) call can be used to set the local address and port number of a UDP socket. The local IP or IPv6 address may be left unspecified in the bind() call by using the special value INADDR_ANY for IP, or the unspecified address (all zeroes) for IPv6. If the bind() call is not done, a local IP or IPv6 address and port number will be assigned to the endpoint when the first packet is sent. Broadcast packets may be sent, assuming the underlying network supports this, by using a reserved “broadcast address.” This address is network interface dependent. Broadcasts may only be sent by the privileged user.

IPv6 does not support broadcast addresses; their function is supported by IPv6 multicast addresses.

Options at the IP level may be used with UDP; see ip(7P) or ip6(7p).

There are a variety of ways that a UDP packet can be lost or corrupted, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its high-water capacity are silently discarded.
UDP processes Internet Control Message Protocol ("ICMP") and Internet Control Message Protocol Version 6 ("ICMP6") error messages received in response to UDP packets it has sent. See \texttt{icmp(7P)} and \texttt{icmp6(7P)}.

ICMP "source quench" messages are ignored. ICMP "destination unreachable," "time exceeded" and "parameter problem" messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

ICMP6 "destination unreachable" packets are ignored unless the enclosed code indicates that the port is not in use on the target host, in which case, the application is notified. ICMP6 "parameter problem" notifications are similarly passed upstream. All other ICMP6 messages are ignored.

\textbf{SEE ALSO}\begin{itemize}
\item \texttt{read(2)}, \texttt{write(2)}, \texttt{bind(3SOCKET)}, \texttt{connect(3SOCKET)}, \texttt{recv(3SOCKET)}, \texttt{send(3SOCKET)}, \texttt{icmp(7P)}, \texttt{icmp6(7P)}, \texttt{inet(7P)}, \texttt{inet6(7P)}, \texttt{ip(7P)}, \texttt{ip6(7P)}, \texttt{tcp(7P)}
\end{itemize}


\textbf{DIAGNOSTICS}\begin{itemize}
\item \texttt{EISCONN} A \texttt{connect()} operation was attempted on a socket on which a \texttt{connect()} operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
\item \texttt{EISCONN} A \texttt{sendto()} or \texttt{sendmsg()} operation specifying an address to which the message should be sent was attempted on a socket on which a \texttt{connect()} operation had already been performed.
\item \texttt{ENOTCONN} A \texttt{send()} or \texttt{write()} operation, or a \texttt{sendto()} or \texttt{sendmsg()} operation not specifying an address to which the message should be sent, was attempted on a socket on which a \texttt{connect()} operation had not already been performed.
\item \texttt{EADDRINUSE} A \texttt{bind()} operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
\item \texttt{EADDRNOTAVAIL} A \texttt{bind()} operation was attempted on a socket with a network address for which no network interface exists.
\item \texttt{EINVAL} A \texttt{sendmsg()} operation with a non-NULL \texttt{msg_accrights} was attempted.
\end{itemize}
### UDP Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EACCES</strong></td>
<td>A <code>bind()</code> operation was attempted with a “reserved” port number and the effective user ID of the process was not the privileged user.</td>
</tr>
<tr>
<td><strong>ENOBUS</strong></td>
<td>The system ran out of memory for internal data structures.</td>
</tr>
</tbody>
</table>
NAME | uhci – host controller driver
SYNOPSIS | pcivid,pid@unit-address
DESCRIPTION | The uhci host controller driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the Universal Host Controller Interface Specification 1.1, an industry standard developed by Intel. The uhci driver supports all USB transfers, including interrupt, control, isochronous and bulk.

The uhci driver supports the nexus device control interface.

FILES | /kernel/drv/uhci 32-bit 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>Limited to PCI-based X86 systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb</td>
</tr>
</tbody>
</table>

SEE ALSO | hubd(7D), usba(7D), usb_mid(7D)

Writing Device Drivers

Universal Host Controller Interface Specification for USB 1.1

Universal Serial Bus Specification 1.0 & 1.1

System Administration Guide: Basic Administration

DIAGNOSTICS | None.
usba(7D)

NAME     usba – Solaris USB Architecture (USBA)

DESCRIPTION USB provides a low-cost means for attaching peripheral devices, including mass-storage devices, keyboards, mice, and printers, to a system. For complete information on USB, go to the USB website at http://www.usb.org.

USB supports 126 hot-pluggable USB devices per USB bus. The maximum data transfer rate is 12 Mbits per second (Mbps).

USB adheres to the Universal Serial Bus 1.1 specification and provides a transport layer abstraction to USB client drivers.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO hid(7D), hubd(7D), ohci(7D), scsa2usb(7D), uhci(7D), usb_mid(7D), usbprn(7D), usbms(7M)

Universal Serial Bus Specification 1.0 and 1.1.

System Administration Guide: Basic Administration

DIAGNOSTICS The messages described below may appear on the system console as well as being logged. All messages are formatted in the following manner:

WARNING: Error message...

<naminumber>: obsolete driver:

usb_pipe_policy is <version> expecting <actual_version>

The driver is using an older revision of USBA. The pipe policy revision used is older and this driver is not supported on the current platform. <name><number> refer to the driver name and its instance number, respectively.

No driver found for device <device_name> (interface <number>

node name=<node_name>)

The installed Solaris software does not contain a supported driver for this hardware. <number> is the interface number.

No driver found for device <name>.

The installed Solaris software does not contain a supported driver for this hardware. <name> could be the device path name or the device name.

Onlining <path name> failed (<number>).

616  man pages section 7: Device and Network Interfaces • Last Revised 28 Sep 2000
usba(7D)

The USB device driver could not be brought online due to internal kernel errors. 
<number> is return value returned due to the failure.
usb_ac(7D)

**NAME**
usb_ac – USB audio control driver

**SYNOPSIS**
usb_ac@unit-address

**DESCRIPTION**
The `usb_ac` driver is a USB (Solaris USB Architecture) compliant client driver that supports the USB Audio Class 1.0 specification.

The audio control driver is a class driver that offers functionality similar to the `audiocs` (sun4u) and `audiots` (Sun Blade 100) drivers which use the Solaris audio mixer framework (mixer(7I)). Unlike the `audiocs` and `audiots` drivers, the USB audio driver may have play-only or record-only capability.

Drivers corresponding to other USB audio interfaces, including the `usb_as(7D)` audio streaming driver or the `hid(7D)` driver, are plumbed under the USB audio control driver and do not directly interface with user applications.

The `usb_ac` driver supports USB audio class compliant devices with a feature unit. For a list of recommended devices, visit: www.sun.com/io.

This interface is described in the `mixer(7I)` and `audio(7I)` man pages.

Applications that open /dev/audio may use the AUDIO_GETDEV ioctl() to determine which audio device is being used. The USB audio driver returns the string "USB Audio" in the name field of the audio_device structure. The version field displays the version number and the config field displays the string "external."

The USB audio device provides support for an external speaker and microphone.

The configuration file /kernel/drv/usb_ac.conf is used to configure the USB audio driver and determines whether the audio mixer is enabled or disabled. See the `mixer(7I)` manual page for details. You can change the audio mixer mode at any time by using the `mixerctl(1)` or `sdtaudiocontrol(1)` applications.

The USB audio device supports the audio data formats shown below. In addition, it must support all of the following sampling frequencies. In the table below, mode "M" indicates that mixer mode is enabled, while "C" indicates that mixer mode is disabled or in compatibility mode.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000 Hz</td>
<td>u-Law or A-Law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>u-law or A-law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>u-law or A-law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>u-law or A-law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>u-law or A-law</td>
<td>8</td>
<td>1 or 2</td>
<td>M and C</td>
</tr>
</tbody>
</table>
As described in the audio(7I) and mixer(7I) man pages, it is possible to request asynchronous notification of changes in the state of an audio device.

If a device is hot-removed while it is active, all subsequent opens will return EIO. All other errors are defined in the audio(7I) man page.

FILES

/kernel/drv/usb_ac
32 bit ELF kernel module.

/kernel/drv/sparcv9/usb_ac
64 bit ELF kernel module.

/kernel/drv/usb_ac.conf
USB audio driver configuration file.

/dev/audio
Symlink to the system’s primary audio device, not necessarily a USB audio device.

/dev/audioctl
/dev/audio control device.

/dev/sound/[0-N]
Represents the audio devices on the system and is not necessarily a USB audio device.

/dev/sound/[0-N].ctl
/dev/sound audio control 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>Limited to PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWuaud, SUNWuaudx</td>
</tr>
</tbody>
</table>
### See Also

mixerctl(1), ioctl(2), attributes(5), ohci(7D), uhci(7D), usb_as(7D), usba(7D), usb_mid(7D), audio(7I), mixer(7I), streamio(7I)

### Writing Device Drivers

*Universal Serial Bus Specification 1.0 and 1.1*

*Universal Serial Bus Device Class Definition for Audio Devices, Release 1.0*

### Diagnostics

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

Warning: <device path> (usb_ac<instance num>): Error Message...

Failure to plumb audio streams drivers.

The usb audio streaming driver (usb_as(7D)) or the hid driver (hid(7D)) could not be plumbed under the audio control driver and the device is not usable.

Device was disconnected while open. Data may have been lost.

The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job will be aborted.

Cannot access device. Please reconnect <name>.

There was an error in accessing the device during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.

A USB audio device was hot-removed while open. A new device was hot-inserted which is not identical to the original USB audio device. Please disconnect the USB device and reconnect the device to the same port.

Busy device has been reconnected.

A device that was hot-removed from a USB port has been re-inserted again.

### Notes

The USB audio device will be power managed if the device is idle.

If the USB audio control driver instance is busy, a system suspend will fail.

USB audio devices do not have line out or port control.
If a USB audio device is hot-removed while active, it prints a console warning message requesting you to put the device back in the same port and informing you that there may be data loss. Hot-removal of an active audio device is strongly discouraged.

Close all applications before hot-removing or hot-inserting a device. If an application is open when a device is hot-removed, inserting the device in a different port will create new /dev/sound links but /dev/audio will not be affected. Hotplugging an active device is not recommended.

On slower IA machines and with higher frequency sample rates, you may encounter some audio performance problems.

To make a USB audio device the primary audio device (for example: /dev/audio), close all audio applications, disconnect all USB audio devices, modunload all other audio drivers and then simply reconnect the USB audio device. This will cause /dev/audio to point to the USB audio /dev/sound entry.

Most Solaris audio applications and third party audio applications available on Solaris work well with USB audio devices. For details of the application behavior with USB audio devices, visit www.sun.com/io.
usb_as(7D)

NAME  usb_as – USB audio streaming driver

SYNOPSIS  usb_as@unit-address

DESCRIPTION  The usb_as driver is a USBA (Solaris USB Architecture) compliant client driver that supports the USB Audio Class 1.0 specification.

The usb_as driver processes audio data messages during play and record and sets sample frequency, precision, encoding and other functions on request from the USB audio control driver. See usb_ac(7D).

This driver is plumbed under the USB audio control driver and does not directly interface with the user application.

FILES  
/kernel/drv/usb_as
   32 bit ELF kernel module

/kernel/drv/sparcv9/usb_as
   64 bit 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>Limited to PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWuaud, SUNWuaudx</td>
</tr>
<tr>
<td>Stability level</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO  mixerctl(1), ioctl(2), attributes(5), ohci(7D), uhci(7D), usb(7D), usb_ac(7D), usb_mid(7D), audio(7I), mixer(7I), streamio(7I)

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

Universal Serial Bus Device Class Definition for Audio Devices, Release 1.0

DIAGNOSTICS  In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

Warning: <device path> (usb_as<instance num>): Error Message...

where <device path> is the physical path to the device in /devices directory.

No bandwidth available.

There is no bandwidth available for the isochronous pipe. As a result, no data will be transferred during play and record.

Cannot access device. Please reconnect <name>.
There was an error in accessing the device during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port.
    Please disconnect and reconnect.

A USB audio streaming interface was hot-removed while open. A new device was hot-inserted which is not identical to the original USB audio device. Please disconnect the USB device and reconnect the device to the same port.

**NOTES**

The USB audio streaming interface will be power managed if device is idle.

If the USB Audio Streaming driver instance is busy, a system suspend will fail.
usbkbm(7M)

NAME  usbkbm – keyboard STREAMS module for Sun USB Keyboard

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

DESCRIPTION  The usbkbm STREAMS module processes byte streams generated by a keyboard attached to a USB port. USB keyboard is a member of Human Interface Device (HID) Class, and usbkbm only supports the keyboard protocol defined in the specification. Definitions for altering keyboard translation and reading events from the keyboard are in \<sys/kbio.h> and \<sys/kbd.h>.

The usbkbm STREAMS module adheres to the interfaces exported by kb(7M). Refer to the DESCRIPTION section of kb(7M) for a discussion of the keyboard translation modes and the IOCTL section for the supported ioctl() requests.

IOCTLs  USB Keyboard usbkbm returns different values for the following ioctls than kb(7M):

  KIOCTYPE  This ioctl() returns a new keyboard type defined for the USB keyboard. All types are listed below:

    KB_SUN3  Sun Type 3 keyboard
    KB_SUN4  Sun Type 4 keyboard
    KB_ASCII  ASCII terminal masquerading as keyboard
    KB_PC  Type 101 PC keyboard
    KB_USB  USB keyboard

    The USB keyboard type is KB_USB; usbkbm will return KB_USB in response to the KIOCTYPE ioctl.

  KIOCLAYOUT  The argument is a pointer to an int. The layout code specified by the bCountryCode value returned in the HID descriptor is returned in the int pointed to by the argument. The countrycodes are defined in 6.2.1 of the HID 1.0 specifications.

  KIOCCMD  KBD_CMD_CLICK/KBD_CMD_NOCLICK  The kb(7M) indicates that inappropriate commands for particular keyboards are ignored. Because clicking is not supported on the USB keyboard, usbkbm ignores this command.

    KBD_CMD_SETLED  Set keyboard LEDs. Same as kb(7M).

    KBD_CMD_GETLAYOUT  The country codes defined in 6.2.1 of the HID 1.0 specification are returned.
KBD_CMD_BELL/KBD_CMD_NOBELL

This command is supported although the USB keyboard does not have a buzzer. The request for the bell is rerouted.

KBD_CMD_RESET

There is no notion of resetting the keyboard as there is for the type4 keyboard. `usbkbm` ignores this command and does not return an error.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO

dumpkeys(1), kbd(1), loadkeys(1), keytables(4), attributes(5), hid(7D), usba(7D), termio(7I), kb(7M)

STREAMS Programming Guide

System Administration Guide: Basic Administration

DIAGNOSTICS

None
usb_mid(7D)

NAME | usb_mid – USB Multi Interface Driver

SYNOPSIS | device@unit-address

DESCRIPTION | The usb_mid driver is a USBA (Solaris Universal Serial Bus Architecture) compliant nexus driver that binds to device level nodes if no vendor or class specific driver is available. The usb_mid driver attempts to bind drivers to each of its interfaces.

The usb_mid driver supports the nexus device control interface.

FILES | /kernel/drv/usb_mid 32-bit ELF kernel module
       /kernel/drv/sparcv9/usb_mid 64-bit 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>PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO | hubd(7D), ohci(7D), usba(7D)

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

System Administration Guide: Basic Administration

DIAGNOSTICS | In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

Warning: <device path> {usb_mid<instance number>}: Error Message...

Cannot access device. Please reconnect <device name>.

This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Device not identical to the previous one on this port. Please disconnect and reconnect.

Same condition as described above; however in this case, the driver is unable to identify the original device with a name string.
NAME
usbms – USB mouse STREAMS module

SYNOPSIS
#include <sys/vuid_event.h>
#include <sys/msio.h>
#include <sys/msreg.h>

DESCRIPTION
The usbms STREAMS module processes byte streams generated by a USB mouse. A USB mouse is a member of the Human Interface Device (HID) class and the usbms module supports only the mouse boot protocol defined in the HID specification. The usbms module must be pushed on top of the HID class driver (see hid(7D)). In the VUID_FIRM_EVENT mode, the usbms module translates packets from the USB mouse into Firm events. The Firm event structure is defined in <sys/vuid_event.h>. The STREAMS module state is initially set to raw or VUID_NATIVE mode which performs no message processing. See the HID 1.0 specification for the raw format of the mouse packets. To initiate mouse protocol conversion to Firm events, change the state to VUID_FIRM_EVENT.

IOCTLS

VUIDGFORMAT
This option returns the current state of the STREAMS module. The state of the usbms STREAMS module may be either VUID_NATIVE (no message processing) or VUID_FIRM_EVENT (convert to Firm events).

VUIDSFORMAT
The argument is a pointer to an int. Set the state of the STREAMS module to the int pointed to by the argument.

typedef struct vuid_addr_probe {
    short base; /* default vuid device addr directed too */
    union {
        short next; /* next addr for default when VUIDSADDR */
        short current; /* current addr of default when VUIDGADDR */
    } data;
} Vuid_addr_probe;

VUIDSADDR
The argument is a pointer to a Vuid_addr_probe structure. VUIDSADDR sets the virtual input device segment address indicated by base to next.

If base does not equal VKEY_FIRST, ENODEV is returned.

VUIDGADDR
The argument is a pointer to a Vuid_addr_probe structure. Return the address of the virtual input device segment indicated by base to current.

If base does not equal VKEY_FIRST, ENODEV is returned.

ioctl() requests for changing and retrieving mouse parameters use the Ms_parms structure:

typedef struct {
    int jitter_thresh;
}
int speed_law;
int speed_limit;
} Ms_parms;

jitter_thresh is the "jitter threshold" of the mouse. Motions fewer than jitter_thresh units along both axes are accumulated and then sent up the stream after 1/12 second.

speed_law indicates whether extremely large motions are to be ignored. If it is 1, a "speed limit" is applied to mouse motions. Motions along either axis of more than speed_limit units are discarded.

MSIOGETPARMS The argument is a pointer to a Ms_Parms structure. The usbms module parameters are returned in the structure.

MSIOSETPARMS The argument is a pointer to a Ms_Parms structure. The usbms module parameters are set according to the values in the structure.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO ioctl(2), hid(7D), usb(7D)
STREAMS Programming Guide
System Administration Guide: Basic Administration

DIAGNOSTICS None
The usbprn driver is a USBA (Solaris USB Architecture) compliant client driver that supports the USB Printer Class 1.0 specification. The usbprn driver supports a subset of the ecpp(7D) parallel port driver functionality. However, unlike the STREAMS-based ecpp driver, usbprn is a character driver.

The usbprn driver supports all USB printer-class compliant printers; however the following devices are recommended:

<table>
<thead>
<tr>
<th>DEVICE NAME</th>
<th>PRINTER CAPABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexmark Optra E310</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark Optra M410</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark Optra T616</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark Optra W810</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark Optra Color45</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Xerox DocuPrint N2125</td>
<td>PostScript printer</td>
</tr>
</tbody>
</table>

The following USB parallel printer adapters are recommended:

<table>
<thead>
<tr>
<th>ADAPTER NAMES</th>
<th>PART NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belkin</td>
<td>F5U002</td>
</tr>
<tr>
<td>Entrega</td>
<td>UP-6</td>
</tr>
</tbody>
</table>

The following parallel port printers are recommended with the above mentioned USB parallel printer adapters:

<table>
<thead>
<tr>
<th>DEVICE NAME</th>
<th>PRINTER CAPABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hewlett Packard LaserJet 6MP</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark Color 45</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Lexmark SC1275</td>
<td>PostScript printer</td>
</tr>
<tr>
<td>Sun SparcE</td>
<td>PostScript printer</td>
</tr>
</tbody>
</table>
The `usbprn` driver supports non-PostScript printers that utilize third-party PostScript conversion packages such as GhostScript. Conversion packages can be obtained from the Solaris 8 Software companion CD, available at http://www.sun.com/software/solaris/binaries/package.html.

With certain minor exceptions (outlined in the Notes sections below), the `usbprn` driver supports a subset of the `ecpp(7D)` ioctl interfaces:

Configuration variables are set to their default values each time the USB printer device is attached. The `write_timeout` period (defined in the ECPIPIOC_SETPARMS ioctl description below) is set to 90 seconds. The mode is set to centronics mode (ECPP_CENTRONICS). Parameters can be changed through the ECPIPIOC_SETPARMS ioctl and read through the ECPIPIOC_GETPARMS ioctl. Each time the USB printer device is opened, the device is marked as busy and all further opens will return EBUSY. Once the device is open, applications can write to the device and the driver can send data and obtain device id and status.

**Note** – Unlike the `ecpp(7D)` driver, `usbprn` resets configuration variables to their default values with each `attach(9E)`. (The `ecpp(7D)` driver resets configuration variables with each `open(2)`.)

A `write(2)` operation returns the number of bytes successfully written to the device. If a failure occurs while a driver is transferring data to printer, the contents of the status bits are captured at the time of the error and can be retrieved by the application program using the ECPIPIOC_GETERR ioctl(2) call. The captured status information is overwritten each time an ECPIPIOC_TESTIO ioctl(2) occurs.

The following ioctl(2) calls are supported.

**ECPIPIOC_GETPARMS**

Gets current transfer parameters. The argument is a pointer to struct `ecpp_transfer_parms`. If parameters are not configured after the device is opened, the structure will be set to its default configuration.

**Note** – Unlike the `ecpp(7D)` driver, only the ECPP_CENTRONICS mode is currently supported in `usbprn`.

**ECPIPIOC_SETPARMS**

Sets transfer parameters. The argument is a pointer to a struct `ecpp_transfer_parms`. If a parameter is out of range, EINVAL is returned. If the peripheral or host device cannot support the requested mode, EPROTONOSUPPORT is returned.

The transfer parameters structure is defined in `<sys/ecppio.h>`.
struct ecpp_transfer_parms {
    int write_timeout;
    int mode;
};

The \texttt{write\_timeout} field, which specifies how long the driver will take to transfer 8192 bytes of data to the device, is set to a default value of 90 seconds. The \texttt{write\_timeout} field must be greater than one second and less than five minutes. All other values are out of range.

\textbf{Note} – Unlike the \texttt{ecpp(7D)} driver, only the ECPP\_CENTRONICS mode is currently supported in \texttt{usbprn}. Also, the semantics of \texttt{write\_timeout} in \texttt{usbprn} differs from \texttt{ecpp(7D)}. Refer to \texttt{ecpp(7D)} for information.

\textbf{BPPIOC\_TESTIO}

Tests the transfer readiness of a print device and checks status bits to determine if a \texttt{write(2)} will succeed. If status bits are set, a transfer will fail. If a transfer will succeed, zero is returned. If a transfer fails, the driver returns \texttt{EIO} and the state of the status bits are captured. The captured status can be retrieved using the \texttt{BPPIOC\_GETERR ioctl(2)} call. \texttt{BPPIOC\_TESTIO} and \texttt{BPPIOC\_GETERR} are compatible to the ioctls specified in \texttt{bpp(7D)}.

\textbf{Note} – Unlike \texttt{ecpp(7D)} driver, only the ECPP\_CENTRONICS mode is currently supported in \texttt{usbprn}. Additionally, \texttt{bus\_error} and \texttt{timeout\_occurred} fields are not used in the \texttt{usbprn} interface. (In \texttt{ecpp(7D)}, \texttt{timeout\_occurred} is used.)

\textbf{BPPIOC\_GETERR}

Get last error status. The argument is a pointer to a \texttt{struct bpp\_error\_status}. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a \texttt{write(2)} call, or the status of the bits at the most recent \texttt{BPPIOC\_TESTIO ioctl(2)} call.

\begin{verbatim}
struct bpp_error_status {
    char timeout_occurred; /* not used */
    char bus_error; /* not used */
    uchar_t pin_status; /* status of pins which
                      /* could cause error */
};
\end{verbatim}

The \texttt{pin\_status} field indicates possible error conditions. The error status structure \texttt{bpp\_error\_status} is defined in the include file \texttt{<sys/bpp\_io.h>}. The valid bits for \texttt{pin\_status} can be \texttt{BPP\_ERR\_ERR}, \texttt{BPP\_SLCT\_ERR}, and \texttt{BPP\_PE\_ERR}. A set bit indicates that the associated pin is asserted.

\textbf{Note} – Unlike the \texttt{ecpp(7D)} driver, only the ECPP\_CENTRONICS mode is currently supported in \texttt{usbprn}. Additionally, the \texttt{bus\_error} and \texttt{timeout\_occurred} fields are not used in the \texttt{usbprn} interface. (In \texttt{ecpp(7D)}, \texttt{timeout\_occurred} is used.) Unlike \texttt{ecpp(7D)}, the \texttt{BPP\_BUSY\_ERR} status bit is not supported by USB printers.
ECPPIOC_GETDEVID

Gets the IEEE 1284 device ID from the peripheral. The argument is a pointer to a
struct ecpp_device_id. Applications should set mode to
ECPP_CENTRONICS. If another mode is used, the driver will return
EPROTONOSUPPORT. len is the length of the buffer pointed to by addr. rlen is the
actual length of the device ID string returned from the peripheral. If the returned
rlen is greater than len, the application should call ECPPIOC_GETDEVID a
second time with a buffer length equal to rlen.

The 1284 device ID stucture:

struct ecpp_device_id {
    int mode; /* mode to use for reading device id */
    int len; /* length of buffer */
    int rlen; /* actual length of device id string */
    char *addr; /* buffer address */
}

Note – Unlike ecpp(7D), only the ECPP_CENTRONICS mode is currently
supported in usbprn.

The read operation is not supported, returns EIO.

EBUSY The device has been opened and another open is
attempted. An attempt has been made to unload
the driver while one of the units is open.

EINVAL An unsupported IOCTL has been received. A
ECPPIOC_SETPARMS ioctl(2) is attempted with an
out of range value in the ecpp_transfer_parms
structure.

EIO The driver has received an unrecoverable device error,
or the device is not responding, or the device has
stalled when attempting an access. A write(2) or
ioctl(2) did not complete due to a peripheral access.
A read(2) system call has been issued.

ENXIO The driver has received an open(2) request for a unit
for which the attach failed.

ENODEV The driver has received an open(2) request for a device
that has been disconnected.

EPROTONOSUPPORT The driver has received a ECPPIOC_SETPARMS
ioctl(2) for a mode argument other than
ECPP_CENTRONICS in the ecpp_transfer_parms
structure.

FILES

/kernel/drv/usbprn
    32 bit ELF kernel module

/kernel/drv/sparcv9/usbprn
    64 bit ELF kernel module
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 PCI-based systems</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusb, SUNWusbx</td>
</tr>
</tbody>
</table>

SEE ALSO

printmgr(1M), bpp(7D), ecpp(7D), hubd(7D), ohci(7D), uhci(7D), usba(7D)

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

USB Device Class Definition for Printing Devices 1.0

System Administration Guide: Basic Administration

DIAGNOSTICS

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

Warning: <device path> (usbprn<instance num>): Error Message...

Device was disconnected while open. Data may have been lost.

The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Cannot access device. Please reconnect <device name>.

There was an error in accessing the printer during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port.
Please disconnect and reconnect.

A USB printer was hot-removed while open. A new device was hot-inserted which is not identical to the original USB printer. Please disconnect the USB device and reconnect the printer to the same port.

Device has been reconnected, but data may have been lost.

The printer that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but the job that was running prior to the hot-removal may be lost.

NOTES

The USB printer will be power managed if the device is closed.
If a printer is hot-removed before a job completes, the job is terminated and the driver will return EIO. All subsequent opens will return ENODEV. If a printer is hot-removed, an LP reconfiguration may not be needed if a printer is re-inserted on the same port. If re-inserted on a different port, an LP reconfiguration may be required.

The USB Parallel Printer Adapter is not hotpluggable. The printer should be connected to USB Parallel Printer Adapter before plugging the USB cable into host or hub port and should be removed only after disconnecting the USB cable of USB Parallel Printer Adapter from the host or hub port.
### NAME
uscsi – user SCSI command interface

### SYNOPSIS
```c
#include <sys/scsi/impl/uscsi.h>

ioctl(int fi, 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:

```c
int uscsi_flags;                     /* read, write, etc. see below */
short uscsi_status;                 /* resulting status */
short uscsi_timeout;                /* Command Timeout */
caddr_t uscsi_cdb;                  /* CDB to send to target */
caddr_t uscsi_bufaddr;              /* i/o source/destination */
size_t uscsi_buflen;                /* size of i/o to take place*/
size_t uscsi_resid;                 /* resid from i/o operation */
uchar_t uscsi_cdblen;               /* # of valid CDB bytes */
uchar_t uscsi_rqlen;                /* size of uscsi_rqbuf */
uchar_t uscsi_rqstatus;             /* status of request sense cmd */
uchar_t uscsi_rqresid;              /* resid of request sense cmd */
caddr_t uscsi_rqbuf;                /* request sense buffer */
void *uscsi_reserved_5;             /* Reserved for future use */
```

The fields of the `uscsi_cmd` structure have the following meanings:

- **uscsi_flags**: The I/O direction and other details of how to carry out the SCSI command. Possible values are described below.
- **uscsi_status**: The SCSI status byte returned by the device is returned in this field.
- **uscsi_timeout**: Time in seconds to allow for completion of the command.
uscsi(7I)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uscsi_cdb</td>
<td>A pointer to the SCSI CDB (command descriptor block) to be transferred to the device in command phase.</td>
</tr>
<tr>
<td>uscsi_bufaddr</td>
<td>The user buffer containing the data to be read from or written to the device.</td>
</tr>
<tr>
<td>uscsi_buflen</td>
<td>The length of uscsi_bufaddr.</td>
</tr>
<tr>
<td>uscsi_resid</td>
<td>If a data transfer terminates without transferring the entire requested amount, the remainder, or residue, is returned in this field.</td>
</tr>
<tr>
<td>uscsi_cdblen</td>
<td>The length of the SCSI CDB to be transferred to the device in command phase.</td>
</tr>
<tr>
<td>uscsi_rqlen</td>
<td>The length of uscsi_rqbuf, the application's Request Sense buffer.</td>
</tr>
<tr>
<td>uscsi_rqstatus</td>
<td>The SCSI status byte returned for the Request Sense command executed automatically by the driver in response to a Check Condition status return.</td>
</tr>
<tr>
<td>uscsi_rqresid</td>
<td>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).</td>
</tr>
<tr>
<td>uscsi_rqbuf</td>
<td>Points to a buffer in application address space to which the results of an automatic Request Sense command are written.</td>
</tr>
<tr>
<td>uscsi_reserved_5</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

The uscsi_flags field defines the following:

- USCSI_WRITE /* send data to device */
- USCSI_SILENT /* no error messages */
- USCSI_DIAGNOSE /* fail if any error occurs */
- USCSI_ISOLATE /* isolate from normal commands */
- USCSI_READ /* get data from device */
- USCSI_ASYNC /* set bus to asynchronous mode */
- USCSI_SYNC /* return bus to sync mode if possible */
- USCSI_RESET /* reset target */
- USCSI_RESET_ALL /* reset all targets */
- USCSI_RQENABLE /* enable request sense extensions */

The uscsi_flags bits have the following interpretation:

- USCSI_WRITE Data will be written from the initiator to the target.
- USCSI_SILENT The driver should not print any console error messages or warnings regarding failures associated with this SCSI command.
**USCSI_DIAGNOSE**  The driver should not attempt any retries or other recovery mechanisms if this SCSI command terminates abnormally in any way.

**USCSI_ISOLATE**  This SCSI command should not be executed with other commands.

**USCSI_READ**  Data will be read from the target to the initiator.

**USCSI_ASYNC**  Set the SCSI bus to asynchronous mode before running this command.

**USCSI_SYNC**  Set the SCSI bus to synchronous mode before running this command.

**USCSI_RESET**  Send a SCSI Bus Device Reset Message to this target.

**USCSI_RESET_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:
SUNWhea

**SEE ALSO**
ioctl(2), attributes(5), sd(7D), st(7D)

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

**WARNINGS**
The `uscsi` command is very powerful, but somewhat dangerous, and so its use is restricted to processes running as root, regardless of the file permissions on the device node. The device driver code expects to own the device state, and `uscsi` commands can change the state of the device and confuse the device driver. It is best to use `uscsi` commands only with no side effects, and avoid commands such as Mode Select, as they may cause damage to data stored on the drive or system panics. Also, as the commands are not checked in any way by the device driver, any block may be overwritten, and the block numbers are absolute block numbers on the drive regardless of which slice number is used to send the command.
usoc – universal serial optical controller for Fibre Channel arbitrated loop (SOC+)
device driver

DESCRIPTION
The Fibre Channel adapter is an SBus card that implements two full duplex Fibre
Channel interfaces. Each interface can connect to a Fibre Channel arbitrated loop
(FC-AL). The usoc device driver is a nexus driver and implements portions of the FC-2
and FC-4 layers of FC-AL.

FILES
/kernel/drv/usoc
32-bit ELF kernel module
/kernel/drv/sparcv9/usoc
64-bit ELF kernel module

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
<tr>
<td>Interface stability</td>
<td>Unknown</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWusoc</td>
</tr>
</tbody>
</table>

SEE ALSO
fctl(7D), sbus(4), fcp(7D), fp(7D), ssd(7D)

Writing Device Drivers

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994
Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996
Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) NCITS TR-19:1998
Fabric Channel Loop Attachment (FC-FLA), NCITS TR-20:1998

DIAGNOSTICS
The following messages are logged and may also appear on the system console. On
the console these messages are preceded by:

usoc%d:

where

usoc%d:
is the per-port instance number of the usoc controller.

Fibre Channel is ONLINE

The Fibre Channel loop is now online.

Fibre Channel Loop is ONLINE
The Fibre Channel loop is now online.

Fibre Channel Loop is OFFLINE

The Fibre Channel loop is now offline.

attach failed: device in slave-only slot.

Move soc+ card to another slot.

attach failed: alloc soft state.

Driver did not attach, devices will be inaccessible.

attach failed: bad soft state.

Driver did not attach, devices will be inaccessible.

attach failed: unable to map eeprom

Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to map XRAM

Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to map registers

Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to access status register

Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to install interrupt handler

Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to access host adapter XRAM

Driver was unable to access device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: unable to write host adapter XRAM

Driver was unable to write device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

attach failed: read/write mismatch in XRAM
Driver was unable to verify device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
visual io(7I)

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 ioctl()s 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 ioctl()s 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 ioctl()s 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 ioctl()s.

IOCTLs

VIS_GETIDENTIFIER
This ioctl() returns an identifier string to uniquely identify a device used in the Solaris VISUAL environment. This is a mandatory ioctl() and must return a unique string. We suggest that the name be formed as <companysymbol><devicetype>. For example, the cgsix driver returns SUNWcg6.

VIS_GETIDENTIFIER takes a vis_identifier structure as its parameter. This structure has the form:

#define VIS_MAXNAMELEN 128
struct vis_identifier {
    char name[VIS_MAXNAMELEN];
};

VIS_GETCURSOR
VIS_SETCURSOR
These ioctl()s fetch and set various cursor attributes, using the vis_cursor structure.

struct vis_cursorpos {
    short x; /* cursor x coordinate */
    short y; /* cursor y coordinate */
};

struct vis_cursorcmap {
    int version; /* version */
    int reserved;
    unsigned char *red; /* red color map elements */
    unsigned char *green; /* green color map elements */
    unsigned char *blue; /* blue color map elements */
};
#define VIS_CURSOR_SETCURSOR 0x01 /* set cursor */
#define VIS_CURSOR_SETPOSITION 0x02 /* set cursor position */
#define VIS_CURSOR_SETHOTSPOT 0x04 /* set cursor hot spot */
#define VIS_CURSOR_SETCOLOMAP 0x08 /* set cursor colormap */
#define VIS_CURSOR_SETSHAPE 0x10 /* set cursor shape */
#define VIS_CURSOR_SETALL (VIS_CURSOR_SETCURSOR | VIS_CURSOR_SETPOSITION | VIS_CURSOR_SETHOTSPOT | VIS_CURSOR_SETCOLORMAP | VIS_CURSOR_SETSHAPE)

struct vis_cursor {
    short set; /* what to set */
    short enable; /* cursor on/off */
    struct vis_cursorpos pos; /* cursor position */
    struct vis_cursorpos hot; /* cursor hot spot */
    struct vis_cursorcmap cmap; /* color map info */
    struct vis_cursorpos size; /* cursor bitmap size */
    char *image; /* cursor image bits */
    char *mask; /* cursor mask bits */
};

The `vis_cursorcmap` structure should contain pointers to two elements, specifying the red, green, and blue values for foreground and background.

VIS_SETCURSORPOS
VIS_MOVECURSOR
These `ioctl()`'s fetch and move the current cursor position, using the `vis_cursorpos` structure.

The following set of `ioctl()`'s are used by graphics drivers that are to be part of the system console device. All of the `ioctl()`'s 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 `ioctl()`'s (listed below) must also be implemented.

The coordinate system for the console device places 0,0 at the upper left corner of the device, with rows increasing toward the bottom of the device and columns increasing from left to right.

VIS_PUTCMAP
VIS_GETCMAP
Set or get color map entries.

The argument is a pointer to a `vis_cmap` structure, which contains the following fields:

struct vis_cmap {
    int index;
    int count;
    uchar_t *red;
    uchar_t *green;
    uchar_t *blue;
};

Device and Network Interfaces 643
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.

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:
### struct vis_display

```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 checked 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`.

- **fg_color** and **bg_color**: fields define the foreground and background color map indexes to use when displaying the data. `fb_color` is used for "on" pixels and `bg_color` is used for "off" pixels.

### VIS_CONSCOPY

Copy data from one location on the device to another. The driver is expected to copy the specified data. The source data should not be modified. Any modifications to the source data should be as a side effect of the copy destination overlapping the copy source.

The argument is a pointer to a `vis_copy` structure which contains the following fields:
struct vis_copy {
    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 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 ioctl()s 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 ioctl()s, 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. It 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. It 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. It takes in as an argument a vis_copy structure.
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>
<tr>
<td>/vol/dev/rdiskette0/unlabeled</td>
<td>unlabeled floppy-raw device access</td>
</tr>
<tr>
<td>/vol/dev/dsk/c0t6/unnamed_cdrom</td>
<td>CD-ROM-block device access</td>
</tr>
</tbody>
</table>
For more information on the location of CD-ROM and floppy media, see *System Administration Guide, Volume 1* or `rmmount(1M)`.

### Partitions

Some media support the concept of a partition. If the label identifies partitions on the media, the name of the media becomes a directory with partitions under it. Only valid partitions are represented. Partitions cannot be moved out of a directory.

For example, if disk volume ‘foo’ has three valid partitions, 0, 2, and 5, then:

```
/vol/dsk/foo/s0
/vol/dsk/foo/s2
/vol/dsk/foo/s5
```

for block access and

```
/vol/rdsk/foo/s0
/vol/rdsk/foo/s2
/vol/rdsk/foo/s5
```

for character access.

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)`, `vold(1M)`, `rmmount.conf(4)`, `vold.conf(4)`

*Solaris Transition Guide*

*System Administration Guide, Volume 1*
NAME
vuídmc3, vuídmc4, vuídmc5, vuídmc2ps, vuídmc3ps – converts mouse protocol to Firm Events

SYNOPSIS
#include <sys/stream.h>
#include <sys/vuid_event.h>

int ioctl(fd, I_PUSH, vuídmc3p);
int ioctl(fd, I_PUSH, vuídmc4p);
int ioctl(fd, I_PUSH, vuídmc5p);
int ioctl(fd, I_PUSH, vuídmc2ps);
int ioctl(fd, I_PUSH, vuídmc3ps);

DESCRIPTION
The STREAMS modules vuídmc3, vuídmc4, vuídmc5, vuídmc2ps, and vuídmc3ps 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:

int format;
format = VUID_FIRM_EVENT;
ioctl(fd, VUIDSPFORMAT, &format);

The user can also query the state of the STREAMS module by using the VUIDGFORMAT option.

int format;
int fd; /* file descriptor */
ioctl(fd, VUIDGFORMAT, &format);
if ( format == VUID_NATIVE )
   /* The state of the module is in raw mode.
   * Message processing is not enabled.
   */
if ( format == VUID_FIRM_EVENT )
   /* Message processing is enabled.
   * Mouse protocol conversion to Firm events
   * are performed.
   */

The remainder of this section describes the processing of STREAMS messages on the read- and write-side.

Read Side
Behavior

M_DATA The messages coming in are queued and converted to Firm events.

M_FLUSH The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.
M_IOCTL

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.

- **VUIDGFORMAT**: This option returns the current state of the STREAMS module. The state of the vuidmice STREAMS module may either be VUID_NATIVE (no message processing) or VUID_FIRM_EVENT (convert to Firm events).

- **VUIDSFORMAT**: This option sets the state of the STREAMS module to VUID_FIRM_EVENT. If the state of the STREAMS module is already in VUID_FIRM_EVENT 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*).

**M_FLUSH**: The write queue of the module is flushed of all its data messages and the message is passed downstream.

### Write Side Behavior

#### Mouse Configurations

<table>
<thead>
<tr>
<th>Module</th>
<th>Protocol Type</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>vuidm3p</td>
<td>3-Byte Protocol Microsoft 2 Button Serial Mouse</td>
<td>/dev/tty*</td>
</tr>
<tr>
<td>vuidm4p</td>
<td>4-Byte Protocol Logitech 3 Button Mouseman</td>
<td>/dev/tty*</td>
</tr>
<tr>
<td>vuidm5p</td>
<td>Logitech 3 Button Bus Mouse Microsoft Bus Mouse</td>
<td>/dev/logi/ dev/msm</td>
</tr>
<tr>
<td>vuid2ps2</td>
<td>PS/2 Protocol 2 Button PS/2 Compatible Mouse</td>
<td>/dev/kdmouse</td>
</tr>
<tr>
<td>vuid3ps2</td>
<td>PS/2 Protocol 3 Button PS/2 Compatible Mouse</td>
<td>/dev/kdmouse</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>IA</td>
</tr>
</tbody>
</table>

### SEE ALSO

attributes(5)
### NAME
wrsm – WCI Remote Shared Memory (WRSM) device driver

### SYNOPSIS
-wci@<slot>,0:wrsm
-wrsm@<instance>:ctrl
-wrsm@ffff,0:admin

### DESCRIPTION
The wrsm driver is a nexus driver that manages WCI devices and wrsm controllers.

A WCI device on a WCI Interface Board (WIB) sits directly on the host backplane and provides clustering communication between Solaris instances that are memory transaction-based. The WCI replaces a schizo or CPU in Sun Fire servers and acts as a memory controller on the system backplane. The wrsm driver programs registers on the WCI to accept network read/write requests on certain exported cluster addresses from incoming links. The registers translate the requests into local read/write bus transactions that use local physical memory ranges that you specify. The driver programs additional WCI registers to forward local system backplane read/write transactions within a particular physical address range to a remote WCI. A WCI device in the format wci@<slot>,0:wrsm appears in the device tree.

A wrsm controller is a pseudo device that manages a set of WCIs. A device entry in the format wrsm@<instance>:ctrl appears in the device tree. A wrsm controller presents a Sun proprietary protocol to clients, enabling them to set up the network and to communicate through the WCIs. To configure a wrsm controller, you download a configuration into the driver using the wrsmconf(1M) command or through other external WCI network management software. Status information on each WCI and wrsm controller is available by using the wrsmstat(1M) command.

The wrsm admin device is used internally by the driver to manage the I/O addresses associated with remote memory. A device entry in the format wrsm@ffff,0:admin appears in the device tree.

### FILES
/platform/sun4u/kernel/drv/sparcv9/wrsm
ELF kernel module

### SEE ALSO
wrsmconf(1M), wrsmstat(1M)

### Writing Device Drivers

### DIAGNOSTICS
The messages described below may appear on the system console as well as being logged. These messages generally include the string wrsm%d, where %d is the instance number of the wrsm device. The message context indicates whether the device is a WCI or a wrsm controller. Some messages include the string wci %a, where %a is the bus slot of the WCI device.

**wrsm%d: unable to map register set %d**

Driver was unable to map device registers; check for bad hardware. Driver did not attach device, device will be inaccessible.

**wrsm_detach: cf_remove_controller failed for wrsm%d**

Last Revised 17 Jan 2001
Driver did not detach device; device is inaccessible.

wrsm_detach: cf_remove_wci failed for wrsm%d

Driver did not detach device. This WCI is the last WCI in wrsm controller.

register_controller of wrsm%d failed with error %d

The wrsm controller could not register with the Sun proprietary protocol framework. Communication is not possible through this controller.

wrsm%d, wci %a, SRAM CE ERROR, at address: 0x%x, syndrome: 0x%x

There was a correctable error in the WCI's SRAM. This indicates that the memory on this WCI module should be replaced.

wrsm%d, wci %a, SRAM UE ERROR, at address: 0x%x, syndrome: 0x%x

There was an uncorrectable error in the WCI's SRAM. This indicates that the memory on this WCI module should be replaced. In addition, attempts to access local memory from remote nodes may fail.
NAME
wrsmd – WCI Remote Shared Memory (WRSM) DLPI driver

SYNOPSIS
wrsmd

DESCRIPTION
The wrsmd device driver is a pseudo driver that presents a type II DLPI interface. The driver uses a Sun proprietary interface over the wrsm driver to provide IP-based communication over the WCI network.

Before using DLPI over a WCI network, you must first configure the wrsm controller on that network. See wrsmconf(1M) for more information. Each wrsmd device is associated with a wrsm controller with a matching id.

FILES
/platform/sun4u/kernel/drv/sparcv9/wrsmd
ELF kernel module

SEE ALSO
wrsconf(1M), wrsm(7D)

Writing Device Drivers
NAME
wscons – workstation console

SYNOPSIS
#include <sys/strredir.h>

ioctl(fd, SRIOCSREDIR, target);
ioctl(fd, SRIOCISREDIR, target);

DESCRIPTION
The wscons workstation console consists of a workstation keyboard and frame buffer
that act together to emulate an ASCII terminal. It includes a redirection facility that
allows I/O issued to the workstation console to be diverted to a STREAMS device,
enabling window systems to redirect output that would otherwise appear directly on
the frame buffer in corrupted form.

Redirection
The wscons redirection facility maintains a list of devices that are designated as
redirection targets through the SRIOCSREDIR ioctl described below. Only the current
entry is active; when the active entry is closed, the most recent remaining entry
becomes active. The active entry acts as a proxy for the device being redirected and
handles all read(2), write(2), ioctl(2), and poll(2) calls issued against the
redirectee.

The ioctls described below control the redirection facility. In both cases, fd is a
descriptor for the device being redirected (or workstation console) and target is a
descriptor for a STREAMS device.

SRIOCSREDIR Designates target as the source and destination of I/O
ostensibly directed to the device denoted by fd.

SRIOCISREDIR Returns 1 if target names the device currently acting as
proxy for the device denoted by fd, and 0 if it is not.

ANSI Standard
Terminal Emulation
On SPARC systems, the PROM monitor emulates an ANSI X3.64 terminal.

On IA systems, the Solaris console subsystem provides ANSI X3.64 emulation.

Note: The VT100 adheres the ANSI X3.64 standard. However, because the VT100
features nonstandard extensions to ANSI X3.64, it is incompatible with Sun terminal
emulators.

The SPARC console displays 34 lines of 80 ASCII characters per line. The IA console
displays 25 lines of 80 ASCII characters per line. Devices with smaller text capacities
may display less. On SPARC systems, the screen-#rows screen-#columns
should be set to 34 or 80 respectively, or text capacities will vary from those described
above. On SPARC systems, the screen-#rows and screen-#columns fields are
stored in NVRAM/EEPROM. See eeprom(1M) for more information. Both SPARC
and IA consoles offer scrolling, (x, y) cursor addressing ability, and a number of other
control functions.

The console cursor marks the current line and character position on the screen. ASCII
characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters.
When a print character is written to the console (and is not part of an escape
sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line.

On SPARC based systems, later PROM revisions have the full 8-bit ISO Latin-1 (ISO 8859-1) character set. Earlier PROM revisions display characters in the range 0xA0 through 0xFE as spaces.

When the cursor is at the right edge of the screen, it moves to the first character position on the next line. When the cursor is at the screen’s right-bottom edge, the line-feed function is performed (see CTRL-J below). The line-feed function scrolls the screen up by one or more lines before moving the cursor to the first character position on the next line.

The wscons console defines a number of control sequences that may occur during input. When a control sequence is written to the console, it affects one of the control functions described below. Control sequences are not displayed on screen.

A number of control sequences (or control character functions) are of the form:

CTRL-\(x\)

where \(x\) represents a single character, such as CTRL-\(J\) for a line feed.

Other ANSI control sequences are of the form:

\(\text{ESC} \ [ \text{params} \ \text{char}\)

**Note** – Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

**ESC** ASCII escape character (ESC, CTRL-\[, 0x1B).

\([\text{Left square bracket}\) ‘\(\text{[}\)’(0x5B).

**params** Sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

Parameters are represented by \(n\) in the syntax descriptions for escape sequence functions.

**char** Function character, which is different for each control sequence and it represented by \(x\) in the syntax descriptions for control character functions.

In the following examples of syntactically valid escape sequences, ESC represent the single ASCII character, Escape:

ESC\(m\) Select graphic rendition with default parameter

ESC\(7m\) Select graphic rendition with reverse image

ESC\(33;54\text{H}\) Set cursor position
Syntactically valid control characters and ANSI escape sequences that are not currently interpreted by the console are ignored.

Each control function requires a specified number of parameters. If fewer parameters are supplied, the remaining parameters (which certain exceptions that are noted below) default to 1. For example, if more than the required number of parameters are supplied, only the last \( n \) is used, where \( n \) is the number required by that particular command character.

Parameters which are omitted or set to 0 are reset to the default value of 1 (with certain exceptions). For example, the command character \( M \) requires one parameter. \( ESC[;M, ESC[0M, ESC[M \) and \( ESC[23;15;32;1M \) are all equivalent to \( ESC[1M \) and provide a parameter value of 1. Note that \( ESC[;5M \) (interpreted as \( \text{‘}ESC[5M\text{’} \)) is not equivalent to \( ESC[5;M \) (interpreted as \( \text{‘}ESC[5;1M\text{’} \)) which is ultimately interpreted as \( \text{‘}ESC[1M\text{’} \)).

The following paragraphs specify the ANSI control functions implemented by the console. Each description provides:
- Control sequence syntax
- Hexadecimal equivalent of control characters where applicable
- Control function name and ANSI or Sun abbreviation (if any).
- Description of parameters required, if any
- Description of the control function
- Initial setting of the mode for functions that set a mode. To restore the initial settings, use the SUNRESET escape sequence.

The wscons control character functions are:

**Bell (BEL)**,
CTRL-G

0x7

Used for consoles that are not equipped with an audible bell. Current Sun workstation models also flash the screen if the keyboard is not the console input device.

**Backspace (BS)**,
CTRL-H,

0x8

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, no change takes place.

**Tab (TAB)**,
CTRL-I,

0x9
The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of eight columns. If the cursor is already at the right edge of the screen, nothing change takes place. Otherwise, the cursor moves right a minimum of one and a maximum of eight character positions.

Line-feed (LF),
CTRL-J,
0xA
The cursor, while remaining at the same character position on the line, moves down one line. If the cursor is at the bottom line, the screen either scrolls up or wraps around depending on the setting of an internal variable $n$ (initially 1). The internal variable can be changed using the $\text{ESC}\left[ x \right]$ control sequence. If $n$ is greater than zero, the entire screen (including the cursor) is scrolled up by $n$ lines before executing the line-feed. The top $n$ lines scroll off the screen and are lost. New blank lines $n$ scroll onto the bottom of the screen. After scrolling, move the cursor down one line to execute the line feed.

If $n$ is zero, wrap-around mode is entered. The $\text{ESC}\left[ 1 x \right]$ exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. During line-feeds, the line that the cursor moves to is cleared and no scrolling occurs. Wrap-around mode is not implemented in the window system.

On SPARC based systems, the speed at which the screen scrolls is dependent on the amount of data waiting to be printed. Whenever a scroll occurs and the console is in normal scroll mode ($\text{ESC}\left[ 1 x \right]$), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when the console finds a control character from the set \{VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US\}. At that point, the screen is scrolled by $n$ lines ($n \geq 1$) and processing continues. The scanned text is processed normally and fills in the newly created lines. As long as escape codes or other control characters are not intermixed with the text, this results in faster scrolling.

Reverse Line-feed,
CTRL-K,
0xB
While remaining at the same character position on the line, the cursor moves up one line. If the cursor is already at the top line, no change takes place.

Form-feed (FF)
CTRL-L,
0xC
The cursor is positioned to the home position (upper-left corner) and the entire screen is cleared.

Return (CR),
CTRL-M,
0xD
The cursor moves to the leftmost character position on the current line.
The `wscons` escape sequence functions are:

- **Escape (ESC),**
  - **CTRL-[,**
  - 0x1B
    - The escape character. Escape initiates a multi-character control sequence.

- **Insert Character (ICH)**
  - `ESC[@`
    - Takes one parameter, \(n\) (default 1). Inserts \(n\) spaces at the current cursor position. The current line, starting at the current cursor position inclusive, is shifted to the right by \(n\) character positions to make room for the spaces. The rightmost \(n\) character positions shift off the line and are lost. The position of the cursor is unchanged.

- **Cursor Up (CUU),**
  - `ESC[A`
    - Takes one parameter, \(n\) (default 1). Moves the cursor up \(n\) lines. If the cursor is fewer than \(n\) lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

- **Cursor Down (CUD),**
  - `ESC[B`
    - Takes one parameter, \(n\) (default 1). Moves the cursor down \(n\) lines. If the cursor is fewer than \(n\) lines from the bottom of the screen, moves the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

- **Cursor Forward (CUF),**
  - `ESC[C`
    - Takes one parameter, \(n\) (default 1). Moves the cursor to the right by \(n\) character positions on the current line. If the cursor is fewer than \(n\) positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

- **Cursor Backward (CUB),**
  - `ESC[D`
    - Takes one parameter, \(n\) (default 1). Moves the cursor to the left by \(n\) character positions on the current line. If the cursor is fewer than \(n\) positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

- **Cursor Next Line (CNL),**
  - `ESC[E`
    - Takes one parameter, \(n\) (default 1). Positions the cursor at the leftmost character position on the \(n\)-th line below the current line. If the current line is less than \(n\) lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

- **Horizontal and Vertical Position (HVP),**
  - `ESC[1;#f`
    - or
Cursor Position (CUP),
ESC[#1;#2H
   Takes two parameters, \textit{n}1 and \textit{n}2 (default 1, 1). Moves the cursor to the \textit{n}2-th character position on the \textit{n}1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.

Erase in Display (ED),
ESC[J
   Takes no parameters. Erases from the current cursor position inclusive to the end of the screen, that is, to the end of the current line and all lines below the current line. The cursor position is unchanged.

Erase in Line (EL),
ESC[K
   Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

Insert Line (IL),
ESC[#L
   Takes one parameter, \textit{n} (default 1). Makes room for \textit{n} new lines starting at the current line by scrolling down by \textit{n} lines the portion of the screen from the current line inclusive to the bottom. The \textit{n} new lines at the cursor are filled with spaces; the bottom \textit{n} lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

Delete Line (DL),
ESC[#M
   Takes one parameter, \textit{n} (default 1). Deletes \textit{n} lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by \textit{n} lines. The \textit{n} new lines scrolling onto the bottom of the screen are filled with spaces; the \textit{n} old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

Delete Character (DCH),
ESC[#P
   Takes one parameter, \textit{n} (default 1). Deletes \textit{n} characters starting with the current cursor position. Shifts the tail of the current line to the left by \textit{n} character positions from the current cursor position, inclusive, to the end of the line. Blanks are shifted into the rightmost \textit{n} character positions. The position of the cursor on the screen is unchanged.

Select Graphic Rendition (SGR),
ESC[#m
   Takes one parameter, \textit{n} (default 0). Note that unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered
according to the parameter until the next occurrence of this escape sequence in the
data stream. Currently only two graphic renditions are defined:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal rendition</td>
</tr>
<tr>
<td>7</td>
<td>Negative (reverse) image</td>
</tr>
</tbody>
</table>

Negative image displays characters as white-on-black if the screen mode is currently black-on-white, and vice-versa. Any non-zero value of \( n \) is currently equivalent to 7 and selects the negative image rendition.

On IA systems only, the following ISO 6429-1983 graphic rendition values support color text:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>black foreground</td>
</tr>
<tr>
<td>31</td>
<td>red foreground</td>
</tr>
<tr>
<td>32</td>
<td>green foreground</td>
</tr>
<tr>
<td>33</td>
<td>brown foreground</td>
</tr>
<tr>
<td>34</td>
<td>blue foreground</td>
</tr>
<tr>
<td>35</td>
<td>magenta foreground</td>
</tr>
<tr>
<td>36</td>
<td>cyan foreground</td>
</tr>
<tr>
<td>37</td>
<td>white foreground</td>
</tr>
<tr>
<td>40</td>
<td>black background</td>
</tr>
<tr>
<td>41</td>
<td>red background</td>
</tr>
<tr>
<td>42</td>
<td>green background</td>
</tr>
<tr>
<td>43</td>
<td>brown background</td>
</tr>
<tr>
<td>44</td>
<td>blue background</td>
</tr>
<tr>
<td>45</td>
<td>magenta background</td>
</tr>
<tr>
<td>46</td>
<td>cyan background</td>
</tr>
<tr>
<td>47</td>
<td>white background</td>
</tr>
</tbody>
</table>

Black On White (SUNBOW),

ESC[p

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. This comprises the initial setting of the screen mode on reset.

White On Black (SUNWOB),

ESC[q
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 black on white.

**ESC\[#r**

Set Scrolling (SUNSCRL)

Takes one parameter, \( n \) (default 0). Sets to \( n \) an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of jump when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates wrap mode instead of scrolling. If a linefeed occurs on the bottom line during wrap mode, the cursor goes to the same character position in the top line of the screen. When a line feed occurs, the line that the cursor moves to is cleared and no scrolling occurs. **ESC \[1r** exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

**ESC\[s**

Reset terminal emulator (SUNRESET)

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are unchanged.

### RETURN VALUES

When there are no errors, the redirection ioctl's have return values as described above. Otherwise, they return \(-1\) and set **errno** to indicate the error. If the target stream is in an error state, **errno** is set accordingly.

If the target stream is in an error state, **errno** is set accordingly.

### ERRORS

- **EBADF** target does not denote an open file.
- **ENOSTR** target does not denote a STREAMS device.

### FILES

- **/dev/wscons** Workstation console, accessed via the redirection facility
- **/dev/systty** Devices that must be opened for the **SRIOCSREDIR** and **SRIOCISREDIR** ioctl's.
- **/dev/syscon** Access system console
- **/dev/console** Access system 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>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

**SEE ALSO**

cvc(1M), eeprom(1M), ioctl(2), poll(2), read(2), write(2), cvc(7D), console(7D)

**WARNINGS**
The redirection ioctl block while there is I/O outstanding on the device instance being redirected. If you try to redirect the workstation console while there is a outstanding read, the workstation console will hang until the read completes.

**NOTES**
The cvc facility supersedes the SunOS wscons facility and should not be used with wscons.
xd(7D)

<table>
<thead>
<tr>
<th>NAME</th>
<th>xd, xdc – disk driver for Xylogics 7053 SMD Disk Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>xdc@6d, ee80/xd@slave, 0:partition</td>
</tr>
<tr>
<td></td>
<td>xdc@6d, ee90/xd@slave, 0:partition</td>
</tr>
<tr>
<td></td>
<td>xdc@6d, eea0/xd@slave, 0:partition</td>
</tr>
<tr>
<td></td>
<td>xdc@6d, eeb0/xd@slave, 0:partition</td>
</tr>
</tbody>
</table>

**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 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/cXdYsZ` block devices, controller X, unit Y, slice Z
- `/dev/rdsk/cXdYsZ` raw devices, controller X, unit Y, slice Z

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:
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.
xmemfs – extended memory file system

#include <sys/mount.h>

mount(special_file, directory, MS_DATA, "xmemfs", dataptr, datalen);

The xmemfs file system is an extended memory file system that provides an efficient mechanism for managing and accessing physical memory that exceeds 4 Gbytes in size. Currently, the xmemfs file system is supported on IA32 architecture systems only.

The Physical Address Extension (PAE) is the xmemfs internal processor feature that enables a 36-bit physical memory address that supports up to 64 Gbytes of physical memory. Once mounted, the xmemfs file system provides standard file operations and semantics on directories and regular files only. Because xmemfs does not allow execute permissions to be set on regular files, execution of object files is prevented.

With xmemfs, the special_file argument, (typically the device on which file systems reside), is ignored and serves only as a placeholder. File data and metadata in xmemfs are always memory-resident. The dataptr argument must (at a minimum) contain the required size specific option. See mount_xmemfs(1M) for more information.

Because xmemfs is a memory-based file system, files and directories that are created are not persistent across reboots or unmounts.

To mount the xmemfs file system, do the following: mount -F xmemfs -osize=4g xmem directory

You can also mount a xmemfs file system on /xmem at multi-user startup time prior to physical memory becoming fragmented. To do this, add the following line to your /etc/vfstab file: xmem /xmem xmemfs yes largebsize,size=4g

The xmemfs file system is expressly designed for performance-driven applications (for example, RDBMS) that require large amounts of physical memory. The xmemfs file system provides file system semantics to manage and access extended memory spaces that exceed 4 Gbytes. From an application perspective, extended memory under the control of a mounted xmemfs file system is viewed as a single, large memory pool that can be partitioned as needed through file creation. You can obtain windows into each memory partition by using mmap(2).

Memory controlled by xmemfs can be partitioned by creating files of the required size in the file system. The xmemfs file system allocates sufficient block-sized memory pages for a file based on the file's size. Files can be created using any standard file utility, including mkfile(1M) and dd(1M). The xmemfs file system optimizes the creation of large files that initially contain all zeroes by allocating memory pages for the file 'hole' that is created by writing beyond the end of file.

If sufficient xmemfs extended memory is available, an application can quickly create an 8 Gbyte file in the xmemfs file system by using llseek(2) to offset 8GB-1 and then write(2) a one-byte buffer containing zero. With xmemfs, you can share and protect
partitioned memory by setting appropriate file permissions. To avoid wasting memory resources, (especially with the `-largebsize` option specified), newly created option-specified files should be a multiple of the block size of the `xmemfs` file system. Creation of many small files is strongly discouraged. See `statvfs(2)` for information on determining file system block sizes.

The `xmemfs` file system should only be used with performance-driven applications that require quick access to large amounts of physical memory. Using `xmemfs` for other applications may result in non-optimal use of system resources and possible system performance degradation.

To maximize `xmemfs` ability to access a file’s extended memory partition, use `mmap(2)`. The initial `mmap(2)` call enables the system to assign a map size containing as much memory as an application may actively access at any time. The map size is constrained by the application’s virtual address space, (usually a maximum of 3 Gbytes on machines with more than 4 Gbytes of physical memory). To access extended memory that is not contained in the existing mapping, use `mmap(2)` with the `-MAP_FIXED` flag to remap a window within the address range returned by the initial `mmap` call.

**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>i386</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`df(1M), mount(1M), mount_xmemfs(1M), mmap(2), statvfs(2), mount(2), umount(2), vfstab(4)`

**DIAGNOSTICS**

If the `xmemfs` file system runs out of space, the following message is displayed in the console indicating that there is insufficient memory to satisfy a `write(2)` request:

```
directory: File system full, no memory
```

**WARNINGS**

Files and directories on an `xmemfs` file system are not preserved across reboots or unmounts.
NAME
xt – driver for Xylogics 472 1/2 inch tape controller

SYNOPSIS
xt@2d,ee60:[l,m] [b] [n]
x@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
/dev/rmt/[0-1] [l,m] [b] [n]
raw devices
For raw devices, the density (low, medium), and 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)`, `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  

xy@2d,ee40/xy@slave,0:partition

xy@2d,ee48/xy@slave,0:partition

DESCRIPTION  

The driver for Xylogics 450/451 devices consists of several components: a controller driver module (xyc) 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 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  

/kernel/drv/xyc  driver module
/kernel/drv/xy  driver module
/kernel/drv/xyc.conf  driver configuration file
/kernel/drv/xy.conf  driver configuration file
/dev/dsk/cXdYsZ  block device, controller X, unit Y, slice Z
/dev/rdsk/cXdYsZ  raw device, controller X, unit Y, slice Z

ATTRIBUTES  

See attributes(5) for descriptions of the following attributes:
### ATTRIBUTE TYPE

<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

1seek(2), read(2), write(2), driver.conf(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.
<table>
<thead>
<tr>
<th>NAME</th>
<th>zero – source of zeroes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>A zero special file is a source of zeroed unnamed memory.</td>
</tr>
<tr>
<td></td>
<td>Reads from a zero special file always return a buffer full of zeroes. The file is of infinite length.</td>
</tr>
<tr>
<td></td>
<td>Writes to a zero special file are always successful, but the data written is ignored.</td>
</tr>
<tr>
<td></td>
<td>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 <code>sysconf</code>. Multiple processes can share such a zero special file object provided a common ancestor mapped the object <code>MAP_SHARED</code>.</td>
</tr>
<tr>
<td>FILES</td>
<td><code>/dev/zero</code></td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>fork(2), mmap(2), sysconf(3C)</code></td>
</tr>
</tbody>
</table>
zs – Zilog 8530 SCC serial communications driver

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/term/n", mode);
open("/dev/tty/n", mode);
open("/dev/cua/n", mode);

The Zilog 8530 provides two serial input/output channels capable of supporting a variety of communication protocols. A typical system uses two or more of these devices to implement essential functions, including RS-423 ports (which also support most RS-232 equipment), and the console keyboard and mouse devices.

The zs module is a loadable STREAMS driver that provides basic support for the Zilog 8530 hardware and basic asynchronous communication support. The driver supports the termio(7I) device control functions specified by flags in the c_cflag word of the termios structure and by the IGNBRK, IGNPAR, FPARMX, or INPCK flags in the c_iflag word. All other termio(7I) functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the ldterm(7M) and ttcompat(7M) STREAMS modules are automatically pushed on top of the stream, providing the standard termio(7I) interface.

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

Valid name space entries are /dev/cua/[a-z], /dev/term/[a-z] and /dev/tty[a-z]. The number of entries used in a name space are machine dependent.

The /dev/tty[n] device names only exist if the SunOS 4.x Binary Compatibility Package is installed. The /dev/tty[n] device names are created by the ucblinks command, which is available only with the SunOS 4.x 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 is available that is controlled by the minor device number. By accessing character-special devices with names of the form /dev/cua/[n], it is possible to open a port without the Carrier Detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

Once a /dev/cua/[n] line is opened, the corresponding tty line cannot be opened until the /dev/cua/n line is closed. A blocking open will wait until the /dev/cua/[n] line is closed (which will drop Data Terminal Ready, and Carrier Detect) and carrier is detected again. A non-blocking open will return an error. If the tty line has been opened successfully (usually only when carrier is recognized on the modem), the corresponding /dev/cua/[n] line cannot be opened. This allows a modem to be attached to /dev/term/[n] (renamed from /dev/tty[n]) and used for
dial-in (by enabling the line for login in /etc/inittab) and also used for dial-out (by tip(1) or uucp(1C)) as /dev/cua/[n] when no one is logged in on the line.

### IOCTLs

The zs module supports the standard set of termio ioctl() calls.

If the CRTSCTS flag in the c_cflag field is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the CRTSCTS flag is clear, the state of CTS has no effect.

If the CRTSXOFF flag in the c_cflag field is set, input will be received only if RTS is high; if RTS is low, input will be frozen. If the CRTSXOFF flag is clear, the state of RTS has no effect.

The termios CRTSCTS (respectively CRTSXOFF) flag and termiox CTXSXON (respectively RTSXOFF) can be used interchangeably.

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

The state of the DCD, CTS, RTS, and DTR interface signals may be queried through the use of the TIOCM_CAR, TIOCM_CTS, TIOCM_RTS, and TIOCM_DTR arguments to the TIOCMGET ioctl command, respectively. Due to hardware limitations, only the RTS and DTR signals may be set through their respective arguments to the TIOCMSET, TIOCMBIS, and TIOCMBIC ioctl commands.

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

When the driver is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default. A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. The Alternate Break sequence can be used to remedy this.

Due to a risk of incorrect sequence interpretation, binary protocols such as PPP, SLIP, and others should not be run over the serial console port when Alternate Break sequence is in effect. By default, the Alternate Break sequence is three characters: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For more information on breaking (entering the debugger or monitor, see kbd(1) and kb(7M).

### ERRORS

An open will fail under the following conditions:

- **ENXIO**: The unit being opened does not exist.
- **EBUSY**: The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY  The port is in use by another serial protocol.
EBUSY  The unit has been marked as exclusive-use by another process
       with a TIOCEXCL ioctl() call.
EINTR  The open was interrupted by the delivery of a signal.
FILES
/dev/cua/[a-z]  dial-out tty lines
/dev/term/[a-z]  dial-in tty lines
/dev/tty[a-z]  binary compatibility package device names

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

<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  kadb(1m), tip(1), ucblinks(1B), cu(1C), uucp(1C), ports(1M), ioctl(2), open(2),
           attributes(5), zsh(7D), termio(7I),ldterm(7M), ttcompat(7M), kb(7M),
           ldterm(7M)

SunOS 4.x Binary Compatibility Guide

DIAGNOSTICS  zsn  : silo overflow.
       The Zilog 8530 character input silo overflowed before it could be serviced.

zsn  : 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/zshu, 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 zshu devices provide what is known as a data path which supports the transfer of data via read(2) and write(2) system calls, as well as ioctl(2) calls. Data path opens are exclusive in order to protect against injection or diversion of data by another process.

The zsh device provides a separate control path for use by programs that need to configure or monitor a connection independent of any exclusive access restrictions imposed by data path opens. Up to three control paths may be active on a particular serial channel at any one time. Control path accesses are restricted to ioctl(2) calls only; no data transfer is possible.

When used in synchronous modes, the Z8530 SCC supports several options for clock sourcing and data encoding. Both the transmit and receive clock sources can be set to be the external Transmit Clock (TRxC), external Receive Clock (RTxC), the internal Baud Rate Generator (BRG), or the output of the SCC’s Digital Phase-Lock Loop (DPLL).

The Baud Rate Generator is a programmable divisor that derives a clock frequency from the PCLK input signal to the SCC. A programmed baud rate is translated into a 16-bit time constant that is stored in the SCC. When using the BRG as a clock source the driver may answer a query of its current speed with a value different from the one specified. This is because baud rates translate into time constants in discrete steps, and reverse translation shows the change. If an exact baud rate is required that cannot be obtained with the BRG, an external clock source must be selected.

Use of the DPLL option requires the selection of NRZI data encoding and the setting of a non-zero value for the baud rate, because the DPLL uses the BRG as its reference clock source.

A local loopback mode is available, primarily for use by the syncloop(1M) utility for testing purposes, and should not be confused with SDLC loop mode, which is not supported on this interface. Also, an auto-echo feature may be selected that causes all incoming data to be routed to the transmit data line, allowing the port to act as the remote end of a digital loop. Neither of these options should be selected casually, or left in use when not needed.
The zsh driver keeps running totals of various hardware generated events for each channel. These include numbers of packets and characters sent and received, abort conditions detected by the receiver, receive CRC errors, transmit underruns, receive overruns, input errors and output errors, and message block allocation failures. Input errors are logged whenever an incoming message must be discarded, such as when an abort or CRC error is detected, a receive overrun occurs, or when no message block is available to store incoming data. Output errors are logged when the data must be discarded due to underruns, CTS drops during transmission, CTS timeouts, or excessive watchdog timeouts caused by a cable break.

**IOCTLS**

The zsh driver supports several ioctl() commands, including:

- **S_IOCGETMODE**
  
  Return a struct scc_mode containing parameters currently in use. These include the transmit and receive clock sources, boolean loopback and NRZI mode flags and the integer baud rate.

- **S_IOCSETMODE**
  
  The argument is a struct scc_mode from which the SCC channel will be programmed.

- **S_IOCGETSTATS**
  
  Return a struct sl_stats containing the current totals of hardware-generated events. These include numbers of packets and characters sent and received by the driver, aborts and CRC errors detected, transmit underruns, and receive overruns.

- **S_IOCCLRSTATS**
  
  Clear the hardware statistics for this channel.

- **S_IOCGETSPEED**
  
  Returns the currently set baud rate as an integer. This may not reflect the actual data transfer rate if external clocks are used.

- **S_IOCGETMCTL**
  
  Returns the current state of the CTS and DCD incoming modem interface signals as an integer.

The following structures are used with zsh ioctl() commands:

```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 */
};
```

Device and Network Interfaces  679
### ERRORS

An open() will fail if a STREAMS message block cannot be allocated, or:

- **ENXIO** The unit being opened does not exist.
- **EBUSY** The device is in use by another serial protocol.

An ioctl() will fail if:

- **EINVAL** An attempt was made to select an invalid clocking source.
- **EINVAL** The baud rate specified for use with the baud rate generator would translate to a null time constant in the SCC's registers.

### FILES

- `/dev/zsh[0-1]`, `/dev/zsh` character-special devices
- `/usr/include/sys/ser_sync.h` header file specifying synchronous serial communication definitions

### ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>IA</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.
zsh(7D)

zsh u: invalid operation for clone dev.
   An inappropriate STREAMS message type was passed through a control path. Only
   M_IOCTL and M_PROTO message types are permitted.

zsh u: not initialized, can’t send message
   An M_DATA message was passed to the driver for a channel that had not been
   programmed at least once since the driver was loaded. The SCC’s registers were in
   an unknown state. The S_IOCSETMODE ioctl command performs the programming
   operation.

zsh u: transmit hung
   The transmitter was not successfully restarted after the watchdog timer expired.
Index

Numbers and Symbols
1/2-inch tape drive
   xt — Xylogics 472, 670
1394 OpenHCI host controller driver —
   hci1394, 201
24-bit UPA color frame buffer and graphics
   accelerator — ffb, 180
3COM EtherLink III Ethernet device driver —
   elx, 139
3COM EtherLink III PCMCIA Ethernet Adapter —
   pcelx, 378
3Com Ethernet device driver — elxl, 141
450 SMD Disk driver — xy, 672
451 SMD Disk driver — xy, 672
472 1/2-inch tape drive — xt, 670
4BSD compatibility module — ttcompat, 597
7053 SMD Disk driver — xd, 666

Acer Laboratories Inc. M5451 audio processor
   interface — audiots, 51
Address Resolution Protocol, See ARP
adp — low-level module for controllers based
   on Adaptec AIC-7870P and AIC-7880P SCSI
   chips, 22
afb — Elite3D graphics accelerator driver, 23
ifb — IFB graphics accelerator driver, 228
Am7990 (LANCE) Ethernet device driver
   — le, 302
   — lebuffer, 302
Am7990 (LANCE) Ethernet device driver (continued)
   — ledma, 302
Am79C940 (MACE) Ethernet device driver —
   qec, 428
Am79C940 (MACE) Ethernet device driver —
   qe, 425
AMD PCnet Ethernet controller device driver —
   pcn, 387
an I2O OS specific module that supports SCSA
   interface
   an I2O, 220
ANSI Layered Console Driver — litem, 327
ANSI standard terminal emulation —
   wscons, 657
arp — Address Resolution Protocol, 24
arp ioctl
   SIOCDARP — delete arp entry, 24
   SIOCGARP — get arp entry, 24
   SIOCSARP — set arp entry, 24
asy — asynchronous serial port driver, 27
asynchronous serial port driver — asy, 27
AT attachment disk driver — ata, 30
ata — AT attachment disk driver, 30
audio — audio device interface, 32
audio device
   Sound Blaster 16/Pro/AWE32 — sbpro, 446
audio_support — audio device independent
   driver interface, 49
audioamd — telephone quality audio
   device, 43
audiocs — Crystal Semiconductor 4231 audio Interface, 45
Audio Data Formats for the Multimedia 4231 Codec, 45
Audio Interfaces, 45
Audio Status Change Notification, 47
Driver Version, 45
Sample Granularity, 47
Setting Audio Mixer Mode, 45

audiots — Acer Laboratories Inc. M5451 audio processor interface, 51

authsha1 — HMAC-SHA-1 Authentication Algorithm Module for IPsec, 56

B
bd — SunButtons and SunDials STREAMS module, 57
bpp — bi-directional parallel port, 59
bufmod — STREAMS Buffer module, 64
built-in mouse device interface — kdmouse, 295
bwtwo — black and white frame buffer, 68

C
cadp — SCSI host bus adapter driver, 70
cadp160 — SCSI host bus adapter driver, 69
CD-ROM — ISO 9660 CD-ROM filesystem — hsfs, 213
cdio — CD-ROM control operations, 73
CDROM control operations — cdio, 73
ce — ce Gigabit-Ethernet device driver, 81
cé Primitives, 82
cgforth — 24-bit color memory frame buffer, 85
cgfour — P4-bus 8-bit color memory frame buffer, 86
cgfourteen — 24-bit color graphics device, 87
cgsix — accelerated 8-bit color frame buffer, 88
cgthree — 8-bit color memory frame buffer, 89
cgtwo — color graphics interface, 90
cmdk — common disk driver, 91
cnft — device driver for Compaq NIC, 92

cogent EM960/EM100 Ethernet controller device driver — dnet, 129
color graphics interface
24-bit color memory frame buffer — cgeight, 85
8-bit color memory frame buffer — cgthree, 89
accelerated 8-bit color frame buffer — cgsix, 88
standard frame buffer — cgtwo, 90
P4-bus 8-bit color memory frame buffer — cgfour, 86
Sun color memory frame buffer — tcx, 559
common disk driver — cmdk, 91
Compaq Smart-2 EISA/PCI and Smart-2SL PCI Array Controller driver — smartii, 492
connections, unique stream
line discipline — conld, 96
conld — line discipline for unique connections, 96
console devices
sysmsg, 552
console
STREAMS-based console interface, 98
converts mouse protocol to Firm Events — vuidmice, 651
vuid2ps2, 651
vuid3ps2, 651
vuidm3p, 651
vuidm4p, 651
vuidm5p, 651
core memory
image — mem, 329
cpqncr — low-level module for Compaq 825 and 875 Controllers, 99
cpr — suspend and resume module, 100
Crystal Semiconductor 4231 audio Interface — audiocs, 45
cvc — virtual console driver, 102
cvcredir — virtual console redirection driver, 103

cogent EM960/EM100 Ethernet controller device driver — dnet, 129
dad — driver for IDE disk devices, 104
Device Statistics Support, 104
Data Link Provider Interface
— dlpi, 123
Davicom Fast Ethernet driver for Davicom
DM9102A — dmfe, 127
dbri — ISDN and audio interface, 107
dbri — Dual Basic Rate ISDN and audio
Interface
Audio Data Formats for BRI Interfaces, 108
Audio Data Formats for the Multimedia
Codec/SpeakerBox, 108
Audio Interfaces, 107
Audio Ports, 110
Audio Status Change Notification, 110
ISDN Interfaces, 107
Sample Granularity, 110
delete arp entry ioctl — SIOCDARP, 24
device driver for Compaq NIC — cnft, 92
device interface
Microsoft Bus Mouse — msm, 346
devices
cgfourteen, 87
disk control operations — dkio, 114
disk driver
— fd — floppy, 170
— Xylogics — xd, 666
— Xylogics — xy, 672
disk quotas — quotactl(), 434
dkio — disk control operations, 114
dlpi — Data Link Provider Interface, 123
dmfe — Davicom Fast Ethernet driver for
Davicom DM9102A, 127
dnet — DEC 21040/21140-based Ethernet
Controllers, 129
DOS
— DOS formatted file system — pcfs, 379
dynamic reconfiguration driver — Sun
Enterprise 10000 and Sun Fire 15K, 131
Sun Enterprise 10000 and Sun Fire 15K dynamic
reconfiguration driver, 131
driver for fast poll on many file descriptors —
poll, 411
driver for IDE disk devices — dad, 104
driver for parallel port — lp, 325
driver for SPARC Storage Array disk devices —
ssd, 499
drivers for floppy disks and floppy disk
controllers — fd, 170
—fdc, 170
drivers
— SCSI tape devices — st, 504
Dual Basic Rate ISDN and audio Interface —
dbri, 107
E
ecpp — bi-directional parallel port, 133
Elite3D graphics accelerator driver — afb, 23
elx — 3COM ETHERLINK III Ethernet device
driver, 139
elx — 3COM EtherLink III Ethernet device
driver
elx Primitives, 139
elxl — 3Com Ethernet device driver, 141
eri — eri Fast-Ethernet device driver, 145
esp — ESP SCSI Host Bus Adapter Driver, 149
ESP SCSI Host Bus Adapter Driver — esp, 149
Ethernet device driver
— SMC EtherPower II 10/100 (9432) Ethernet
device driver — spwr, 498
extended memory file system — xmemfs, 668
F
fas — FAS SCSI Host Bus Adapter Driver, 155
FAS SCSI Host Bus Adapter Driver — fas, 155
fbio — frame buffer operations, 163
fcip — IP/ARP over Fibre Channel datagram
encapsulation driver, 165
fcp — Fibre Channel protocol driver, 168
fctl — Sun Fibre Channel transport library, 169
— fd — drivers for floppy disks and floppy disk
controllers, 170
—fdc — drivers for floppy disks and floppy disk
controllers, 170
FDGETCHANGE — get status of disk
changed, 176
fdio — disk control operations, 176
FDIOGCHAR — get floppy
characteristics, 176
FDIOGCHAR — set floppy characteristics, 176
if_tcp — general properties of Internet Protocol
  network interfaces, 233
  Application Programming Interface, 233
IFB graphics accelerator driver — ifb, 228
inet — Internet protocol family, 243
inet6 — Internet protocol family for Internet
  Protocol version 6, 240
Intel D100 Ethernet device driver — iprb, 257
Intel i82365SL PC Card Interface Controller —
  pcic, 384
Intel Ethernet device driver, Intel — iprb, 257
Internet Control Message Protocol —
  icmp, 223
Internet Protocol — ip, 252
Internet Transmission Control Protocol —
  tcp, 554
Internet User Datagram Protocol — udp, 612
Internet Control Message Protocol for Internet
  Protocol Version 6 — icmp6, 221
Internet Protocol
to Ethernet addresses — arp, 24
Internet protocol family — inet, 243
Internet protocol family for Internet Protocol
  version 6 — inet6, 240
Internet Protocol Security Architecture —
  ipsec, 260
Internet Protocol Version 6 — ip6, 246
Internet Protocol Version 6 — ip6, 246
ioctls for sockets
  SIOCADDR — delete arp entry, 24
ioctls for sockets
  SIOCGARP — get arp entry, 24
ioctls for Internet socket descriptors
  SIOCSARP — set arp entry, 24
ioctls for floppy
  FDEJECT — eject floppy, 176
  FDGETCHAGE — get status of disk
    changed, 176
  FDIOCHART — get floppy
    characteristics, 176
  FDIOCHART — set floppy
    characteristics, 176
ioctls for sockets
  SIOCADDRT — add route, 441
  SIOCDELRRT — delete route, 441
ioctl’s for terminals
  TIOCPKT — set/clear packet mode
    (pty), 423
  TIOCREMOTE — remote input editing, 423
  TIOCSSTART — start output (like
    control-Q), 423
  TIOCSSTOP — stop output (like
    control-S), 423
ip — Internet Protocol, 252
IP/ARP over Fibre Channel datagram
  encapsulation driver — fcip, 165
ip6 — Internet Protocol Version 6, 246
ipd — STREAMS modules and drivers for the
  Point-to-Point Protocol, 415
ipdcm — STREAMS modules and drivers for the
  Point-to-Point Protocol, 415
ipdptp — STREAMS modules and drivers for the
  Point-to-Point Protocol, 415
iprb — D100 Ethernet device driver, 257
ipsec — Internet Protocol Security
  Architecture, 260
isdnio — generic ISDN interface, 266
ISO 9660 — ISO 9660 CD-ROM filesystem —
  hsfs, 213
isp — ISP SCSI Host Bus Adapter Driver, 280
ISP SCSI Host Bus Adapter Driver — isp, 280
ISP2200 Family Fibre Channel host bus adapter
driver — qlc, 433

K
kb — keyboard STREAMS module, 286
  Keyboard Compatibility Mode, 290
  Keyboard Translation Mode, 286
  Keyboard Translation-Table Entries, 286
kdmouse — built-in mouse device
  interface, 295
kernel lock statistics driver — lockstat, 316
kernel packet forwarding database —
  route, 437
kernel statistics driver — kstat, 296
kernel symbols — ksysms, 297
keyboard STREAMS module — kb, 286
keyboard STREAMS module for Sun USB
  Keyboard — usbkbm, 624
kstat — kernel statistics driver, 296

Index 687
**L**

LAN support module — gld, 187
ldterm — line discipline for STREAMS terminal module, 299
le — Am7990 (LANCE) Ethernet device driver, 302
lebuffer — Am7990 (LANCE) Ethernet device driver, 302
ledma — Am7990 (LANCE) Ethernet device driver, 302
line discipline for unique stream connections — connd, 96
llcl — Logical Link Control Protocol Class 1 Driver, 306
lockstat — kernel lock statistics driver, 316
lofi — loopback file driver, 317
lofs — loopback virtual file system, 318
log — interface to STREAMS error logging and event tracing, 320
logi — LOGITECH bus mouse device interface, 324
Logical Link Control Protocol Class 1 Driver — llcl, 306
LOGITECH Bus Mouse device interface — logi, 324
loopback file driver — lofi, 317
loopback transport providers
— tclts, 588
— ticots, 588
— ticotsord, 588
loopback virtual file system — lofs, 318
low-level module for Compaq 825 and Compaq 875 Controllers — cpqncr, 99
low-level module for controllers based on
Adaptec AIC-7870P and AIC-7880P SCSI chips — adp, 22
low-level module for Mylex/BusLogic host bus adapters — flashpt, 181
low-level module for the AMD PCscsi, PCscsi II, PCnet-SCSI, and Qlogic QLA510 PCI-to-SCSI bus adapters — pscscl, 390
lp — driver for parallel port, 325
ltm — ANSI Layered Console Driver, 327

**M**
m64 — frame buffers device driver, 328
magnetic tape interface
— mtio, 348
mem — image of core memory, 329
memory, core
image — mem, 329
memory, zeroed unnamed
source — zero, 674
memory based filesystem — tmpfs, 594
message output collection
msglog, 345
mhd — multihost disk control operations, 330
Automatic Probing, 333
MHIOCENFAILFAST ioctl, 333
Non-shared multihost disks:, 330
Shared Multihost Disks, 331
Microsoft Bus Mouse device interface — msm, 346
mixer — audio mixer device interface, 335
monitor
PROM monitor configuration interface — openprom, 372
monochrome frame buffer — bwtwo, 68
Mouse device interface
LOGITECH Bus Mouse device interface — logi, 324
msglog
message output collection, 345
msm — Microsoft Bus Mouse device interface, 346
mtio — general magnetic tape interface, 348
multihost disk control operations — mhd, 330

**N**
llc2 — llc2 Class II logical link control driver, 309
ncrs — ncrs SCSI host bus adapter driver, 360
cadp — cadp SCSI Host Bus Adapter Driver
Driver Configuration, 70
ncrs — ncrs SCSI Host Bus Adapter Driver
Driver Configuration, 361
dman — Network device driver, 124
Network device driver — dman, 124
Network device driver — scman, 449
scman — Network device driver, 449
network packet routing device — routing, 441
null — null file, 367

O
ocf_escr1 — serial smart card terminal
driver, 368
ocf_ibutton — iButton smart card terminal
driver, 369
ocf_iscr1 — I2C smart card card terminal
driver, 370
ohci — OpenHCI host controller driver, 371
OpenHCI host controller driver — ohci, 371
openprom — PROM monitor configuration
interface, 372

P
packet routing device — routing, 441
packet routing ioctls
SIOCADDRT — add route, 441
SIOCDELRT — delete route, 441
parallel port, bi-directional — bpp, 59
parallel port, bi-directional — ecpp, 133
parallel port
driver for parallel port — lp, 325
pcta — PCMCIA ATA card device driver, 377
pcelx — 3COM EtherLink III PCMCIA Ethernet
Adapter, 378
pcfs — DOS formatted file system, 379
pcic — Intel i82365SL PC Card Interface
Controller, 384
pckt — STREAMS Packet Mode module, 385
PCMCIA ATA card device driver — pcta, 377
PCMCIA memory card nexus driver —
pcmcm, 386
PCMCIA RAM memory card device driver —
pcram, 389
PCMCIA serial card device driver —
pcser, 391
pcmcm — PCMCIA memory card nexus
driver, 386
pcn — AMD PCnet Ethernet controller device
driver, 387
pcram — PCMCIA RAM memory card device
driver, 389
pcscsi — low-level module for the AMD PCscsi,
PCscsi II, PCnet-SCSI, and Qlogic QLA510
PCI-to-SCSI bus adapters, 390
pcser — PCMCIA serial card device
driver, 391
pf_key — security association database, 392
pfmod — STREAMS packet filter module, 400
pipemod — STREAMS pipe flushing
module, 403
Platform Specific Module (PSM) for Tricord
Systems Enterprise Server Models ES3000,
ES4000 and ES5000 — tpf, 596
pln — SPARC Storage Array SCSI Host Bus
adapter driver, 404
PLN SCSI Host Bus Adapter driver — pln, 404
pm — Power Management Driver, 405
Point-to-Point Protocol
— ipd, 415
— ipdcm, 415
— ipdptp, 415
— ppp, 415
— ppp_diag, 415
poll — driver for fast poll on many file
descriptors, 411
Power Management Driver — pm, 405
ppp — STREAMS modules and drivers for the
Point-to-Point Protocol, 415
Operation, 415
ppp_diag — STREAMS modules and drivers
for the Point-to-Point Protocol, 415
PROM
monitor configuration interface —
openprom, 372
pseudo-terminal driver — pty, 422
Pseudo Terminal Emulation module, STREAMS
— ptem, 417
ptem — STREAMS Pseudo Terminal Emulation
module, 417
ptm — STREAMS Buffer module, 418
pts — STREAMS pseudo-tty slave driver, 420
pty — pseudo-terminal driver, 422
Q
qe — Am79C940 (MACE) Ethernet device
driver, 425
qec — Am79C940 (MACE) Ethernet device
driver, 428
qfe — SUNW,qfe Quad Fast-Ethernet device
driver, 429
qfe Primitives, 431
qlc — ISP2200 Family Fibre Channel host bus
adapter driver, 433
quotactl() — disk quotas, 434

R
remote input editing ioctl —
TIOCREMOTE, 423
WCI Remote Shared Memory (WRSM) DLPI
driver — wrsm, 656
rns_smt — Rockwell Station Management
driver, 436
Rockwell 2200 SNAP Streams Driver —
xp, 545
Rockwell Station Management driver —
rns_smt, 436
route — kernel packet forwarding
database, 437
Messages, 438
routing — local network packet routing, 441
routing ioctls
SIOCADDRT — add route, 441
SIOCDELRT — delete route, 441
set/clear
packet mode (pty) ioctl — TIOCPKT, 423
sgen — Generic SCSI device driver, 476
zsh — On-board serial HDLC interface, 468,
678
Shared Memory (WRSM) device driver —
wrsmd, 654
Siemens 82532 ESCC serial communications
driver — se, 465
SIOCDARP — delete arp entry, 24
SIOCGARP — get arp entry, 24
SIOCSARP — set arp entry, 24
slp — Service Location Protocol, 490
smartii — Compaq Smart-2 EISA/PCI and
Smart-2SL PCI Array Controller driver, 492
SMC Ethernet device drivers
spwr — SMC EtherPower II 10/100 (9432)
Ethernet device driver, 498
SMC EtherPower 8432BT Ethernet controller
device driver — dnet, 129
SMC EtherPower II 10/100 (9432) Ethernet
device driver — spwr, 498

S
sbaru — Creative Labs Sound Blaster audio
device, 446
scsa2usb — SCSI to USB bridge host bus
adapter driver, 454
SCSI enclosure services device driver —
se, 471
Class II logical link control driver — llc2, 309
ncrs SCSI host bus adapter driver — ncrs, 360
SCSI host bus adapter driver — cadp160, 69
SCSI host bus adapter driver — cadp, 70
SCSI tape devices
driver — st, 504
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear
SCSI to USB bridge host bus adapter driver —
csaa2usb, 454
SCSI disk and ATAPI/SCSI CD-ROM device
driver — sd, 459
sd — SCSI disk and ATAPI/SCSI CD-ROM
device driver, 459
se — Siemens 82532 ESCC serial
communications driver, 465
security association database — pf_key, 392
serial communications driver — zs, 675
Serial Optical Controller device driver —
soc, 493
Serial Optical Controller for Fibre Channel
Arbitrated Loop (SOC+) device driver —
socal, 495
Serial Parallel Communications driver for SBus
— stc, 516
serial smart card terminal driver —
ocf_escr1, 368
Service Location Protocol — slp, 490
ses — SCSI enclosure services device
driver, 471
set/clear

SMD and IPI disk control operations —
   hdio, 202
SMD disk controller
   Xylogics 450 — xy, 672
   Xylogics 451 — xy, 672
   Xylogics 7053 — xd, 666
soc — Serial Optical Controller Device
   Driver, 493
socal — Serial Optical Controller for Fibre
   Channel Arbitrated Loop (SOC+) device
   driver, 495
sockio — ioctls that operate directly on
   sockets, 497
sockets
   ioctls that operate directly — sockio, 497
Solaris USB architecture — usba, 616
Solaris VISUAL I/O control operations, 642
Sound Blaster 16/Pro/AWE32 audio devices —
   sbpro, 446
SPARC storage Array
   disk devices driver — ssd, 499
   spwr — SMC EtherPower II 10/100 (9432)
      Ethernet device driver, 498
   ssd — driver for SPARC Storage Array disk
      devices, 499
   st — driver for SCSI tape devices, 504
   start output (like control-Q) ioctl —
      TIOCSTART, 423
   stc — Serial Parallel Communications driver for
      SBus, 516
   stop output (like control-S) ioctl —
      TIOCSTOP, 423
STP 4020 PCMCIA Adapter
   STP 4020 PCMCIA Adapter, 528
   stp4020 — STP 4020 PCMCIA Adapter, 528
STREAMS
   interface to error logging — log, 320
   interface to event tracing — log, 320
   line discipline for unique stream connections
      — conld, 96
   loopback transport providers — ticlt, ticots, ticotsord, 588
   On-board serial HDLC interface —
      se_hdlc, 468
   On-board serial HDLC interface — zsh, 678
STREAMS (continued)
   standard terminal line discipline module —
      ldterm, 299
   Transport Interface cooperating module —
      timod, 590
   Transport Interface read/write interface
      module — tirdwr, 592
   V7, 4BSD, XENIX compatibility module —
      ttcompat, 597
STREAMS Administrative Driver — sad, 443
STREAMS-based console interface
   console, 98
STREAMS Buffer module — bufmod, 64
STREAMS Buffer module — ptm, 418
STREAMS ioctl commands — streamio, 529
STREAMS module
   SunButtons and SunDials — bd, 57
STREAMS modules and drivers for the
   Point-to-Point Protocol
   — ipd, 415
   — ipdcm, 415
   — ipdptp, 415
   — ppp, 415
   — ppp_diag, 415
STREAMS Packet Filter Module — pfmod, 400
STREAMS Packet Mode module — pckt, 385
STREAMS pipe flushing module —
   pipemod, 403
STREAMS Pseudo Terminal Emulation module
   — ptm, 417
STREAMS pseudo-tty slave driver — pts, 420
   Sun Fibre Channel transport library — fctl, 169
   SunButtons and SunDials STREAMS module —
   bd, 57
   SUNW,hme Fast-Ethernet device driver —
      hme, 206
   SUNW,qfe Quad Fast-Ethernet device driver —
      qfe, 429
   eri Fast-Ethernet device driver — eri, 145
   suspend and resume module — cpr, 100
   xp — Rockwell 2200 SNAP Streams
      Driver, 545
   symhisl — SCSI Host Bus Adapter Driver, 548
   SCSI Host Bus Adapter Driver — symhisl, 548
   sysmsg
      console devices, 552
      system message routing, 552

Index 691
system message routing
  sysmsg, 552

T
  t1394dcam — IEEE 1394 bus-based digital
camera, 553
tape, magnetic interface
    — mrio, 348
tape drive, 1/2-inch
    — xt — Xylopiccs 472, 670
tape interface — mt, 347
tcp — Internet Transmission Control
  Protocol, 554
tcx — Sun low-range graphics accelerator with
color memory frame buffer, 559
terminial, standard STREAMS
    line discipline module — ldterm, 299
terminal emulation, ANSI — wscons, 657
terminal interface
    — termio, 561
terminal interface, extended
    — termiox, 582
terminal parameters — termiox, 582
termio — general terminal interface, 561
  Canonical mode input processing, 562
  Comparison of the different cases of MIN,
  TIME interaction, 564
  Control Modes, 571
  Default values, 579
  Input modes, 568
  Local modes, 574
  Minimum and Timeout, 577
  Modem disconnect, 566
  Modem lines, 578
  Non-canonical mode input processing, 563
  Output modes, 569
  Special Characters, 565
  Terminal parameters, 567
  Terminal size, 577
  Termio structure, 577
  Writing characters, 565
termiox — extended general terminal
  interface, 582
ticlts — loopback transport provider, 588
ticots — loopback transport provider, 588
ticotsord — loopback transport provider, 588
timod — Transport Interface cooperating
  module, 590
timod — Transport Interface read/write
  interface module, 592
TIOCCKT — set/clear packet mode (pty), 423
TIOCREMOTE — remote input editing, 423
TIOCSTART — start output (like
  control-Q), 423
TIOCSTOP — stop output (like control-S), 423
tmpfs — memory based filesystem, 594
tpf — Platform Specific Module (PSM) for
  Tricord Systems Enterprise Server Models
  ES3000, ES4000 and ES5000, 596
Transport Interface cooperating STREAMS
  module — timod, 590
Transport Interface read/write interface
  STREAMS module — timod, 592
ttcompat — V7, 4BSD and XENIX STREAMS
  compatibility module, 597
tty — controlling terminal interface, 604

U
  uata — IDE Host Bus Adapter Driver, 609
  udfs — universal disk format filesystem, 611
  udp — Internet User Datagram Protocol, 612
  universal disk
    universal disk format filesystem —
    udfs, 611
  universal serial optical controller for Fibre
    Channel arbitrated loop (SOC+) device
    driver — usoc, 639
  unnamed zeroed memory
    source — zero, 674
  usb_ac — USB audio control driver, 618
  usb_as — USB audio control driver, 622
  USB hub driver — hubd, 216
  usb_mid — USB Multi Interface Driver, 626
  USB mouse STREAMS module — usbs, 627
  USB Multi Interface Driver — usb_mid, 626
  usbprn — USB printer class driver, 629
  usba — Solaris USB architecture, 616
  usbkbm — keyboard STREAMS module for Sun
    USB Keyboard, 624
usbkbm — keyboard STREAMS module for Sun IOCTLS, 624
usbms — USB mouse STREAMS module, 627
uscsi — user SCSI command interface, 635
user SCSI command interface — uscsi, 635
usoc — universal serial optical controller for Fibre Channel arbitrated loop (SOC+) device driver, 639

V
V7 compatibility module — ttcompat, 597
virtual console driver — cvc, 102
virtual console redirection driver — cvcredir, 103
volfs — Volume Management file system, 649
Volume Management file system — volfs, 649
vuid2ps2 — converts mouse protocol to Firm Events, 651
vuid3ps2 — converts mouse protocol to Firm Events, 651
vuidm3p — converts mouse protocol to Firm Events, 651
vuidm4p — converts mouse protocol to Firm Events, 651
vuidm5p — converts mouse protocol to Firm Events, 651
vuidmice — converts mouse protocol to Firm Events, 651

W
workstation console — wscons, 657
wrsm — Shared Memory (WRSM) device driver, 654
wrsmd — WCI Remote Shared Memory (WRSM) DLPI driver, 656

X
xd — Xylogics SMD Disk driver, 666
XENIX compatibility module — ttcompat, 597
xmemfs — extended memory file system, 668

xt — Xylogics 472 1/2-inch tape drive, 670
xy — Xylogics SMD Disk driver, 672
Xylogics SMD Disk driver — xd, 666
Xylogics SMD Disk driver — xy, 672
Xylogics 472 1/2-inch tape drive — xt, 670

Z
zero — source of zeroes, 674
Zilog 8530 SCC serial communications driver — zs, 675
zs — zilog 8530 SCC serial communications driver, 675