man pages section 7: Device and Network Interfaces

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 835-8007
December 2000

Contents

Preface

Overview

A man page is provided for both the naive user and the sophisticated user who is familiar with the Trusted Solaris operating environment and is in need of online information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Trusted Solaris Reference Manual

In the AnswerBook2™ and online man command forms of the man pages, all man pages are available:

- Trusted Solaris man pages that are unique for the Trusted Solaris environment
- SunOS 5.8 man pages that have been changed in the Trusted Solaris environment
- SunOS 5.8 man pages that remain unchanged.

The printed manual, the *Trusted Solaris 8 Reference Manual* contains:

- Man pages that have been added to the SunOS operating system by the Trusted Solaris environment
- Man pages that originated in SunOS 5.8, but have been modified in the Trusted Solaris environment to handle security requirements.

Users of printed manuals need both manuals in order to have a full set of man pages, since the *SunOS 5.8 Reference Manual* contains the common man pages that are not modified in the Trusted Solaris environment.

Man Page Sections

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

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

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

NAME

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

SYNOPSIS

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

The following special characters are used in this section:

[ ]	The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
. . .	Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ' "`filename` . . ." .
|	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, functions and such, are described under USAGE.

IOCTL

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

OPTIONS

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

OPERANDS

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

OUTPUT

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

RETURN VALUES

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

ERRORS

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

USAGE

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

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be root, `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.

SUMMARY OF TRUSTED SOLARIS CHANGES

This section describes changes to a Solaris item by Trusted Solaris software. It is present in man pages that have been modified from Solaris software.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications. The references are divided into two sections, so that users of printed manuals can easily locate a man page in its appropriate printed manual.

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.

CHAPTER

Device and Network Interfaces

10

NAME

Intro – Introduction to special files

DESCRIPTION

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

This section contains the following major collections:

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

This section contains man pages that are new or modified for Trusted Solaris describing 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, along with any changes in their behavior in the Trusted Solaris operating environment.

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

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

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

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

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

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

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

Trusted Solaris modified and new STREAMS modules are described here. `streamio`(7I) contains a list of ioctl requests used to manipulate STREAMS modules and interface with the STREAMS framework. Ioctl requests specific to a STREAMS module will be discussed on the man page for that module.

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

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

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in `ip`(7P). This is the default protocol for `SOCK_RAW` type sockets within the `AF_INET` domain.
- The Transmission Control Protocol (TCP); see `tcp`(7P). This is the default protocol for `SOCK_STREAM` type sockets.

- The User Datagram Protocol (UDP); see `udp`(7P). This is the default protocol for `SOCK_DGRAM` type sockets.
- The Address Resolution Protocol (ARP); see `arp`(7P).
- The Internet Control Message Protocol (ICMP); see `icmp`(7P).

TRUSTED SOLARIS DIFFERENCES

The Trusted Solaris special files are:

- `SunOS` 5.8 files that have been modified to work within Trusted Solaris security policy, such as `sad`(7D). Man pages for modified special files have been rewritten to remove information that is not accurate for how the special file behaves within the Trusted Solaris environment. Modified man pages also have added descriptions for Trusted Solaris requirements..
- `SunOS` 5.8 special files that remain unchanged from the Solaris release, such as `pipemod`(7M).

The printed *Trusted Solaris 8 Reference Manual* includes only those man pages that have been modified or originate in the Trusted Solaris environment. Printed versions of unchanged `SunOS` 5.8 man pages are found in the *SunOS 5.8 Reference Manual*. See *man pages section 7* for the unchanged pages in AnswerBook2 format.

SEE ALSO

Trusted Solaris 8 Reference Manual

`add_drv`(1M), `rem_drv`(1M), `intro`(3)

SunOS 5.8 Reference Manual

`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, *TCP/IP and Data Communications Administration Guide*, *STREAMS Programming Guide*, *Writing Device Drivers*

Name	Description
`kb`(7M)	Keyboard STREAMS module
`sad`(7D)	STREAMS Administrative Driver
`wscons`(7D)	Workstation console

NAME

kb – Keyboard STREAMS module

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/vuid_event.h>
#include <sys/kbio.h>
#include <sys/kbd.h>
```

ioctl(*fd*, *I_PUSH*, *"kb"*);

DESCRIPTION

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

`kb` recognizes which keys have been typed using a set of tables for each known type of keyboard. Each translation table is an array of 128 16-bit words (`unsigned shorts`). If an entry in the table is less than 0x100, it is treated as an ISO 8859/1 character. Higher values indicate special characters that invoke more complicated actions.

Keyboard Translation Mode

The keyboard can be in one of the following translation modes:

`TR_NONE`	Keyboard translation is turned off and up/down key codes are reported.
`TR_ASCII`	ISO 8859/1 codes are reported.
`TR_EVENT`	`firm_events` are reported.
`TR_UNTRANS_EVENT`	`firm_events` containing unencoded keystation codes are reported for all input events within the window system.

Keyboard Translation-Table Entries

All instances of the `kb` module share seven translation tables used to convert raw keystation codes to event values. The tables are:

Unshifted	Used when a key is depressed and no shifts are in effect.
Shifted	Used when a key is depressed and a Shift key is being held down.
Caps Lock	Used when a key is depressed and Caps Lock is in effect.
Alt Graph	Used when a key is depressed and the Alt Graph key is being held down.

Num Lock	Used when a key is depressed and Num Lock is in effect.
Controlled	Used when a key is depressed and the Control key is being held down (regardless of whether a Shift key or the Alt Graph is being held down, or whether Caps Lock or Num Lock is in effect).
Key Up	Used when a key is released.

Each key on the keyboard has a "key station" code which is a number from 0 to 127. This number is used as an index into the translation table that is currently in effect. If the corresponding entry in that translation table is a value from 0 to 255, this value is treated as an ISO 8859/1 character, and that character is the result of the translation.

If the entry is a value above 255, it is a "special" entry. Special entry values are classified according to the value of the high-order bits. The high-order value for each class is defined as a constant, as shown in the list below. The value of the low-order bits, when added to this constant, distinguishes between keys within each class:

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 key was down.
SYSTEMBIT 1	The System key was pressed. This is a place holder to indicate which key is the system-abort key.

FUNNY 0x300

Performs various functions depending on the value of the low 4 bits:

NOP 0x300	Does nothing.
OOPS 0x301	Exists, but is undefined.
HOLE 0x302	There is no key in this position on the keyboard, and the position-code should not be used.

RESET 0x306	Keyboard reset.
ERROR 0x307	The keyboard driver detected an internal error.
IDLE 0x308	The keyboard is idle (no keys down).
COMPOSE 0x309	This key is the COMPOSE key; the next two keys should comprise a two-character “COMPOSE key” sequence.
NONL 0x30A	Used only in the Num Lock table; indicates that this key is not affected by the Num Lock state, so that the translation table to use to translate this key should be the one that would have been used had Num Lock not been in effect.
0x30B — 0x30F	Reserved for non-parameterized functions.

FA_CLASS 0x400

This key is a “floating accent” or “dead” key. When this key is pressed, the next key generates an event for an accented character; for example, “floating accent grave” followed by the “a” key generates an event with the ISO 8859/1 code for the “a with grave accent” character. The low-order bits indicate which accent; the codes for the individual “floating accents” are as follows:

FA_UMLAUT 0x400	umlaut

	FA_CFLEX 0x401	circumflex
	FA_TILDE 0x402	tilde
	FA_CEDILLA 0x403	cedilla
	FA_ACUTE 0x404	acute accent
	FA_GRAVE 0x405	grave accent

STRING 0x500

The low-order bits index a table of strings. When a key with a `STRING` entry is depressed, the characters in the null-terminated string for that key are sent, character by character. The maximum length is defined as:

KTAB_STRLEN	10

Individual string numbers are defined as:

HOMEARROW	0x00
UPARROW	0x01
DOWNARROW	0x02
LEFTARROW	0x03
RIGHTARROW	0x04

String numbers 0x05 — 0x0F are available for custom entries.

FUNCKEYS 0x600

Function keys. The next-to-lowest 4 bits indicate the group of function keys:

LEFTFUNC	0x600
RIGHTFUNC	0x610
TOPFUNC	0x620
BOTTOMFUNC	0x630

The low 4 bits indicate the function key number within the group:

LF(*n*)	(LEFTFUNC+(*n*)-1)
RF(*n*)	(RIGHTFUNC+(*n*)-1)
TF(*n*)	(TOPFUNC+(*n*)-1)

BF(*n*)	(BOTTOMFUNC+(*n*)-1)

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

PADKEYS 0x700 This key is a "numeric keypad key." These entries should appear only in the Num Lock translation table; when Num Lock is in effect, these events will be generated by pressing keys on the right-hand keypad. The low-order bits indicate which key; the codes for the individual keys are as follows:

PADEQUAL 0x700	"=" key
PADSLASH 0x701	"/" key
PADSTAR 0x702	"*" key
PADMINUS 0x703	"-" key
PADSEP 0x704	"," key
PAD7 0x705	"7" key
PAD8 0x706	"8" key
PAD9 0x707	"9" key
PADPLUS 0x708	"+" key
PAD4 0x709	"4" key
PAD5 0x70A	"5" key
PAD6 0x70B	"6" key
PAD1 0x70C	"1" key
PAD2 0x70D	"2" key
PAD3 0x70E	"3" key
PAD0 0x70F	"0" key
PADDOT 0x710	"." key
PADENTER 0x711	"Enter" key

In `TR_ASCII` mode, when a function key is pressed, the following escape sequence is sent:

ESC[0 . . . . 9z

where ESC is a single escape character and "0...9" indicates the decimal representation of the function-key value. For example, function key R1 sends the sequence:

ESC[208z

because the decimal value of RF(1) is 208. In TR_EVENT mode, if there is a VUID event code for the function key in question, an event with that event code is generated; otherwise, individual events for the characters of the escape sequence are generated.

Keyboard Compatibility Mode

kb is in "compatibility mode" when it starts up. In this mode, when the keyboard is in the TR_EVENT translation mode, ISO 8859/1 characters from the "upper half" of the character set (that is, characters with the 8th bit set) are presented as events with codes in the ISO_FIRST range (as defined in <SYS/VUID_EVENT.H>). The event code is ISO_FIRST plus the character value. This is for backwards compatibility with older versions of the keyboard driver. If compatibility mode is turned off, ISO 8859/1 characters are presented as events with codes equal to the character code.

IOCTLS

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

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

KIOCGTRANS The argument is a pointer to an int. The current translation mode is stored in the int pointed to by the argument.

ioctl( ) requests for changing and retrieving entries from the keyboard translation table use the kiockeymap structure:

```
struct kiockeymap {
 int kio_tablemask; /* Translation table (one of: 0, CAPSMASK,
    * SHIFTMASK, CTRLMASK, UPMASK,
    * ALTGRAPHMASK, NUMLOCKMASK)
    */
#define KIOCABORT1 –1 /* Special "mask": abort1 keystation */
#define KIOCABORT2 –2 /* Special "mask": abort2 keystation */
 uchar_t kio_station; /* Physical keyboard key station (0-127) */
 ushort_t kio_entry; /* Translation table station's entry */
 char kio_string[10]; /* Value for STRING entries (null terminated) */
};
```

KIOCSKEY The argument is a pointer to a kiockeymap structure. The translation table entry referred to by the values in that structure is changed.

kio_tablemask specifies which of the five translation tables contains the entry to be modified:

UPMASK 0x0080	"Key Up" translation table.
NUMLOCKMASK 0x0800	"Num Lock" translation table.
CTRLMASK 0x0030	"Controlled" translation table.
ALTGRAPHMASK 0x0200	"Alt Graph" translation table.
SHIFTMASK 0x000E	"Shifted" translation table.
CAPSMASK 0x0001	"Caps Lock" translation table.
(No shift keys pressed or locked)	"Unshifted" translation table.

kio_station specifies the keystation code for the entry to be modified. The value of kio_entry is stored in the entry in question. If kio_entry is between STRING and STRING+15, the string contained in kio_string is copied to the appropriate string table entry. This call may return EINVAL if there are invalid arguments.

There are a couple special values of kio_tablemask that affect the two step "break to the PROM monitor" sequence. The usual sequence is L1-a or Stop-. If kio_tablemask is KIOCABORT1 then the value of kio_station is set to be the first keystation in the sequence. If kio_tablemask is KIOCABORT2 then the value of kio_station is set to be the second keystation in the sequence.

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 is stored in the int pointed to by the argument, if the keyboard type is unknown. In case of error, -1 is stored in the int pointed to by the argument.

KIOCLAYOUT The argument is a pointer to an int. On a Sun Type 4 keyboard, the layout code specified by the keyboard's DIP switches is stored in the int pointed to by the argument.

KIOCCMD The argument is a pointer to an int. The command specified by the value of the int pointed to by the argument is sent to the keyboard. The commands that can be sent are:

Commands to the Sun Type 3 and Sun Type 4 keyboards:

KBD_CMD_RESET	Reset keyboard as if power-up.
KBD_CMD_BELL	Turn on the bell.
KBD_CMD_NOBELL	Turn off the bell.
KBD_CMD_CLICK	Turn on the click annunciator.
KBD_CMD_NOCLICK	Turn off the click annunciator.

Commands to the Sun Type 4 keyboard:

KBD_CMD_SETLED	Set keyboard LEDs.
KBD_CMD_GETLAYOUT	Request that keyboard indicate layout.

Inappropriate commands for particular keyboard types are ignored. Since there is no reliable way to get the state of the bell or click (because we cannot query the keyboard, and also because a process could do writes to the appropriate serial driver — thus going around this ioctl( ) request) we do not provide an equivalent ioctl( ) to query its state.

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

`LED_KANA`	“Kana” light.

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

`KIOCSCOMPAT` The argument is a pointer to an `int`. “Compatibility mode” is turned on if the `int` has a value of 1, and is turned off if the `int` has a value of 0.

`KIOCGCOMPAT` The argument is a 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` The argument is a pointer to an `int`. The keyboard abort sequence (typically L1-A or Stop-A on the keyboard on SPARC systems and BREAK on the serial console device) effect is enabled if the int has a non-zero value; otherwise, the keyboard abort sequence effect is disabled. When enabled, the default effect causes the operating system to suspend and enter the kernel debugger (if present) or the system PROM (on most systems with OpenBoot PROMs). The default effect is "enabled" on most systems. The default effect may be different on server systems with key switches when the key switch is in the "secure" position. On these server systems, the effect is always "disabled" when the key switch is in the "secure" position. This `ioctl()` request returns `EPERM` if the caller does not have the `PRIV_SYS_DEVICES` privilege in its set of effective privileges.

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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

SUMMARY OF TRUSTED SOLARIS CHANGES

To use the `KIOCSKABORTEN` request, the caller must have `sys_devices` in its set of effective privileges.

SEE ALSO

Trusted Solaris 8 Reference Manual

`kbd`(1), *Trusted Solaris Administrator's Procedures*

SunOS 5.8 Reference Manual

`loadkeys`(1), `kadb`(1M), `keytables`(4), `attributes`(5), `termio`(7I)

NOTES

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

NAME

sad – STREAMS Administrative Driver

SYNOPSIS

#include `<sys/types.h>`

#include `<sys/conf.h>`

#include `<sys/sad.h>`

#include `<sys/stropts.h>`

`int ioctl(int` *fildes*, `int` *command*, `int` *arg*);

DESCRIPTION

The STREAMS Administrative Driver provides an interface for applications to perform administrative operations on STREAMS modules and drivers. The interface is provided through `ioctl`(2) commands. Privileged operations may access the `sad` driver using `/dev/sad/admin`. The requesting process must have `PRIV_SYS_DEVICES` privilege in its effective set. Unprivileged operations may access the `sad` driver using `/dev/sad/user`.

The *fildes* argument is an open file descriptor that refers to the `sad` driver. The *command* argument determines the control function to be performed as described below. The *arg* argument represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a `command`-specific data structure.

COMMAND FUNCTIONS

The autopush facility [see `autopush`(1M)] allows the administrator to configure a list of modules to be automatically pushed on a stream when a driver is first opened. Autopush is controlled by the following commands:

`SAD_SAP` Allows the administrator to configure the given device's autopush information. *arg* points to a `strapush` structure, which contains the following members:

```
unit_t sap_cmd;
major_t sap_major;
minor_t sap_minor;
minor_t sap_lastminor;
unit_t sap_npush;
unit_t sap_list [MAXAPUSH] [FMNAMESZ + 1];
```

The `sap_cmd` field indicates the type of configuration being done. It may take on one of the following values:

`SAP_ONE`	Configure one minor device of a driver.
`SAP_RANGE`	Configure a range of minor devices of a driver.
`SAP_ALL`	Configure all minor devices of a driver.
`SAP_CLEAR`	Undo configuration information for a driver.

The `sap_major` field is the major device number of the device to be configured. The `sap_minor` field is the minor device number of the device to be configured. The `sap_lastminor` field is used only with the `SAP_RANGE` command, which configures a range of minor devices between `sap_minor` and `sap_lastminor`, inclusive. The minor fields have no meaning for the `SAP_ALL` command. The `sap_npush` field indicates the number of modules to be automatically pushed when the device is opened. It must be less than or equal to `MAXAPUSH`, defined in `<sad.h>`. It must also be less than or equal to `NSTRPUSH`, the maximum number of modules that can be pushed on a stream, defined in the kernel master file. The field `sap_list` is an array of `NULL`-terminated module names to be pushed in the order in which they appear in the list.

When using the `SAP_CLEAR` command, the user sets only `sap_major` and `sap_minor`. This will undo the configuration information for any of the other commands. If a previous entry was configured as `SAP_ALL, sap_minor` should be set to zero. If a previous entry was configured as `SAP_RANGE`, `sap_minor` should be set to the lowest minor device number in the range configured.

On failure, `errno` is set to the following value:

`EFAULT`	*arg* points outside the allocated address space.
`EINVAL`	The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.
`ENOSTR`	The major device number does not represent a STREAMS driver.
`EEXIST`	The major-minor device pair is already configured.
`ERANGE`	The command is `SAP_RANGE` and `sap_lastminor` is not greater than `sap_minor`, or the command is `SAP_CLEAR` and `sap_minor` is not equal to the first minor in the range.
`ENODEV`	The command is `SAP_CLEAR` and the device is not configured for autopush.
`ENOSR`	An internal autopush data structure cannot be allocated.

SAD_GAP
: Allows any user to query the sad driver to get the autopush configuration information for a given device. *arg* points to a strapush structure as described in the previous command.

The user should set the sap_major and sap_minor fields of the strapush structure to the major and minor device numbers, respectively, of the device in question. On return, the strapush structure will be filled in with the entire information used to configure the device. Unused entries in the module list will be zero-filled.

On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL The major device number is invalid.

ENOSTR The major device number does not represent a STREAMS driver.

ENODEV The device is not configured for autopush.

SAD_VML
: Allows any user to validate a list of modules (that is, to see if they are installed on the system). *arg* is a pointer to a str_list structure with the following members:

```
int sl_nmods;
 struct str_mlist *sl_modlist;
```

The str_mlist structure has the following member:

```
char    l_name[FMNAMESZ+1];
```

sl_nmods indicates the number of entries the user has allocated in the array and sl_modlist points to the array of module names. The return value is 0 if the list is valid, 1 if the list contains an invalid module name, or –1 on failure. On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL The sl_nmods field of the str_list structure is less than or equal to zero.

SUMMARY OF TRUSTED SOLARIS CHANGES

The PRIV_SYS_DEVICES privilege is required to perform privileged operations.

SEE ALSO

Trusted Solaris 8 Reference Manual

`autopush`(1M), `intro`(2), `open`(2)

STREAMS Programming Guide

SunOS 5.8 Reference Manual

`ioctl`(2), `attributes`(5)

STREAMS Programming Guide

DIAGNOSTICS

Unless otherwise specified, the return value from `ioctl()` is `0` upon success and `−1` upon failure with `errno` set as indicated.

NAME

wscons – Workstation console

SYNOPSIS

#include <sys/strredir.h>
ioctl(*fd*, SRIOCSREDIR, *target*,)
ioctl(*fd*, SRIOCISREDIR, *target*);

DESCRIPTION

The "workstation console" is a device consisting of the combination of the workstation keyboard and frame buffer, acting in concert to emulate an ASCII terminal. It includes a redirection facility that allows I/O issued to the workstation console to be diverted to some other STREAMS device, so that, for example, window systems can arrange to redirect output that would otherwise appear directly on the frame buffer, corrupting its appearance.

Redirection

The redirection facility maintains a list of devices that have been named as redirection targets, through the `SRIOCSREDIR` ioctl described below. All entries but the most recent are inactive; when the currently active entry is closed, the most recent remaining entry becomes active. The active entry acts as a proxy for the device being redirected; it handles all `read`(2), `write`(2), `ioctl`(2), and `poll`(2) calls issued against the redirectee.

The following two ioctls control the redirection facility. In both cases, *fd* is a descriptor for the device being redirected (that is, the workstation console) and *target* is a descriptor for a STREAMS device.

`SRIOCSREDIR`	Make *target* be the source and destination of I/O ostensibly directed to the device denoted by *fd*. The requesting process must have the `PRIV_SYS_CONSOLE` privilege in its effective set for the operation to succeed.
`SRIOCISREDIR`	Returns `1` if *target* names the device currently acting as proxy for the device denoted by *fd*, and `0` if it is not.

SPARC ANSI Standard Terminal Emulation

On SPARC based systems, the PROM monitor emulates an ANSI X3.64 terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun terminal emulator and the VT100 are *not* compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with scrolling, (*x*, *y*) cursor addressability, and a number of other control functions.

While the display size is usually 34 by 80, there are instances where it may be a different size.

- If the display device is not large enough to display 34 lines of text.
- On SPARC based systems, if either screen-#rows or screen-#columns is set by the user to a value other than the default of 34 or 80 respectively. screen-#rows and screen-#columns are fields stored in NVRAM/EEPROM, see `eeprom`(1M).

The Sun console displays a cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line.

On SPARC based systems, later PROM revisions have the full 8-bit ISO Latin-1 (ISO 8859-1) character set, not just ASCII. Earlier PROM revisions display characters in the range 0xA0 – 0xFE as spaces.

If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.

SPARC Control Sequence Syntax

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation CTRL-*X* for some character *X*, represents a control character.

Other ANSI control sequences are of the form `ESC [` *params char*

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC	represents the ASCII escape character (ESC, CTRL-[, 0x1B).
[	The next character is a left square bracket '[' (0x5B).
params	A sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.
char	A function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, `ESC` represent the single ASCII character 'Escape'):

```
ESC[m select graphic rendition with default parameter
ESC[7m select graphic rendition with reverse image
ESC[33;54H set cursor position
ESC[123;456;0;;3;B move cursor down
```

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last *n* are used, where *n* is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as 'ESC[5M') is *not* equivalent to ESC[5;M (interpreted as 'ESC[5;1M') which is ultimately interpreted as 'ESC[1M').

In the syntax descriptions below, parameters are represented as '#' or '`#1;#2`'.

SPARC ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the `SUNRESET` escape sequence.

SPARC Control Character Functions

CTRL-G (0x7)Bell (BEL)

The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It 'rings the bell' by flashing the entire screen. The window system flashes the window. The screen will also be flashed on current models if the Sun keyboard is not the console input device.

CTRL-H (0x8)Backspace (BS)
The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

CTRL-I (0x9)Tab (TAB)
The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

CTRL-J (0xA)Line-feed (LF)

The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or “wraps around” depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, ‘wrap-around’ mode is entered. ‘ESC [1 r’ exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. When any line-feed occurs, the line that the cursor moves to is cleared. This means that no scrolling occurs. Wrap-around mode is not implemented in the window system.

On SPARC based systems, the screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place and the console is in normal scroll mode (‘ESC [1 r’), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when any control character from the set {VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US} is found. At that point, the screen is scrolled by N lines (N ≥ 1) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the ‘Set scrolling’ (ESC[r) control function below.

CTRL-K (0xB)Reverse Line-feed
The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.

CTRL-L (0xC)Form-feed (FF)
The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.

CTRL-M (0xD)Return (CR)
The cursor moves to the leftmost character position on the current line.

SPARC Escape Sequence Functions

CTRL-[(0x1B) Escape (ESC)

This is the escape character. Escape initiates a multi-character control sequence.

ESC[#@ Insert Character (ICH)

Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A Cursor Up (CUU)
Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B Cursor Down (CUD)
Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C Cursor Forward (CUF)
Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D Cursor Backward (CUB)
Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E Cursor Next Line (CNL)
Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f Horizontal And Vertical Position (HVP)
or

ESC[#1;#2H Cursor Position (CUP)
Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.

ESC[J Erase in Display (ED)
Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K Erase in Line (EL)
Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L Insert Line (IL)
Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with spaces; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M Delete Line (DL)
Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with spaces; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P Delete Character (DCH)
Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC[#m Select Graphic Rendition (SGR)
Takes one parameter, # (default 0). Note: Unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0	Normal rendition.
7	Negative (reverse) image

Negative image displays characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and selects the negative image rendition.

On IA systems only, the following ISO 6429-1983 graphic rendition values support color text:

30	black foreground
31	red foreground
32	green foreground
33	brown foreground
34	blue foreground
35	magenta foreground
36	cyan foreground
37	white foreground
40	black background
41	red background
42	green background
43	brown background
44	blue background
45	magenta background
46	cyan background
47	white background

ESC[p Black On White (SUNBOW)
Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see ‘Select Graphic Rendition’ above) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC[q White On Black (SUNWOB)
Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see ‘Select Graphic Rendition’ above) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC[#r Set scrolling (SUNSCRL)
: Takes one parameter, # (default 0). Sets to # an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of “jump” when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates “wrap mode” instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever occurs. ‘ESC [1 r’ exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

ESC[s Reset terminal emulator (SUNRESET)
: Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are unchanged.

RETURN VALUES

When there are no errors, the redirection ioctls have return values as described above. Otherwise, they return `–1` and set `errno` to indicate the error.

If the *target* stream is in an error state, `errno` is set accordingly.

ERRORS

`EPERM` An `SRIOCSREDIR` command is issued, and the requesting process does not have the `PRIV_SYS_CONSOLE` privilege in its effective set.

`EBADF` *target* does not denote an open file.

`ENOSTR` *target* does not denote a STREAMS device.

`EINVAL` (x86 only) *fd* does not denote `/dev/console`.

FILES

`/dev/wscons` The workstation console, accessed by way of the redirection facility

x86 Only

`/dev/systty`
`/dev/syscon`
`/dev/console` The device that must be opened for the `SRIOCSREDIR` and `SRIOCISREDIR` ioctls

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, the `SRIOCSREDIR` command requires the `PRIV_SYS_CONSOLE` privilege. A new error code `EPERM` is added.

SEE ALSO

Trusted Solaris 8 Reference Manual

`write`(2), `read`(2)

SunOS 5.8 Reference Manual

`ioctl`(2), `poll`(2), `attributes`(5), `console`(7D)

WARNINGS

The redirection ioctls block while there is I/O outstanding on the device instance being redirected. Thus, attempting to redirect the workstation console while there is a read outstanding on it will hang until the read completes.

NOTES

On Sun Enterprise 10000 servers the `netcon` facility supersedes `wscons`(7D). `wscons` is useful for systems that do have directly attached consoles, such as frame buffers and keyboards, but it is not useful with the Enterprise 10000 server, which does not. For more information, refer to `netcon`(1M) in the *Sun Enterprise 10000 SSP Reference Manual* or `cvcd`(1M).

Index