Solaris on Sun Hardware Reference Manual Supplement

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No. 817-1872-10
April 2003, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Contents

Preface

The *Solaris on Sun Hardware Reference Manual Supplement* contains reference manual pages (man pages) for software provided to Sun hardware customers with the Solaris 9 product. These supplement the man pages provided in the general *Solaris 9 Reference Manual*. This edition has been updated to include man pages found in the Solaris 9 4/03 release.

Before you can access some of the information published in this book through the `man` command, you may need to install software from the Solaris Software Supplement CD for your Solaris release. In most cases, when you install a software product from the Solaris Software Supplement CD, a package containing man pages about the software will be automatically installed. For information about installing the man page software, refer to the *Solaris 9 Sun Hardware Platform Guide*.

Note – Some man pages delivered on the Supplement CD are published in reference manuals devoted to specific products. Those man pages are not included in the *Solaris on Sun Hardware Reference Manual Supplement*.

How This Book Is Organized

This manual contains man pages in alphabetical order within each category:

- System Administration Commands (1M)
- File Formats (4)
- Device and Network Interfaces (7)

The man pages apply to the following products:

- SunHSI/P™ (PCI bus) network adapter software: `hsip`, `hsip_init`, `hsip_loop`, `hsip_stat`

- Sun Remote System Control (RSC): `rscadm`
- SunVTS™ diagnostic software: `sunvts`, `vts_cmd`, `vtsk`, `vtsprobe`, `vtstty`, `vtsui`
- Netra™ t server environmental monitoring software: `envmond`, `envmond.conf`

Accessing Sun Documentation Online

The `docs.sun.com`SM web site enables you to access Sun technical documentation on the Web. You can browse the archive or search for a specific book title or subject at: `http://docs.sun.com`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at: `docfeedback@sun.com`

Please include the part number (817-1872-10) of your document in the subject line of your email.

NAME

envmond - environmental monitor daemon

SYNOPSIS

/usr/platform/SUNW,UltraSPARC-IIi-Netract/lib/envmond/sparcv9/envmond [**-d**] [**-f** *file*] [**-g** *granularity*]

AVAILABILITY

SUNWcteux

DESCRIPTION

The **envmond** daemon polls system environment monitoring devices to check for conditions that may require corrective action. In order to do this, the daemon reads a configuration file on startup, during the initial Solaris boot process, to find out which environmental devices will be monitored. Each configuration file entry describing an environmental device is referred to as a policy, and the supported policy entries are described in **envmond.conf**(4)**.**

The **envmond** daemon logs appropriate messages to a system log file via **syslogd**(1M)**.**

The **envmond** daemon will reread its configuration information file whenever it receives a hang-up signal, SIGHUP.

OPTIONS

-d Sets Debug mode option. The **envmond** will not run as a daemon, and will instead run in the foreground, inheriting standard input and output. Error and warning messages will be written to the standard output instead of being logged via **syslogd**(1M)**.**

-f *file* Provides an alternate file path for the configuration file.

-g *granularity*
Defines the finest granularity for the poll interval. The default value is 10 seconds.

FILES

/usr/platform/SUNW,UltraSPARC-IIi-Netract/lib/envmond/sparcv9/envmond
The executable daemon

/usr/platform/SUNW,UltraSPARC-IIi-Netract/lib/envmond/sparcv9/*.so
The **envmond** policies

/platform/SUNW,UltraSPARC-IIi-Netract/lib/envmond.conf
The **envmond** configuration file

SEE ALSO

syslogd(1M), **envmond.conf**(4)

NOTES

The **envmond** policies retrieve their environmental information via I2C devices in the system.

This daemon is in the PROTOTYPE stage, and is therefore subject to CHANGE WITHOUT NOTICE.

NAME

hsip_init − set high speed serial line interface operating parameters.

SYNOPSIS

/opt/SUNWconn/bin/hsip_init *device* **[[** *baud_rate* **] | [** *keyword=value***, ...] | [** *single-word option* **]]**

DESCRIPTION

The hsip_init utility allows the user to modify some of the hardware operating modes common to high speed synchronous serial lines. This may be useful in troubleshooting a link, or necessary to the operation of a communications package.

If run without options, hsip_init reports the options as presently set on the port. If options are specified, the new settings are reported after they have been made.

OPTIONS

Options to hsip_init normally take the form of a keyword, followed by an equal sign and a value. The exception is that a baud rate may be specified as a decimal integer by itself. Keywords must begin with the value shown in the options table, but may contain additional letters up to the equal sign. For example, "loop=" and "loopback=" are equivalent.

Recognized options are listed in the table below.

Keyword	Value	Effect
loopback	yes	Set the port to operate in **internal loopback** mode. The receiver is electrically disconnected from the DCE receive data input and tied to the outgoing transmit data line. Transmit data is available to the DCE. If no other clocking options have been specified, perform the equivalent of **txc=baud** and **rxc=baud**.
	no	Disable internal loopback mode. If no other clocking options have been specified, perform the equivalent of **txc=txc** and **rxc=rxc**.
	echo	Set the port to operate in **auto-echo** mode. The port will echo incoming receive data on the transmit data pin. When the loopback is set for echo and no clocking option is given the clocking is set txc=txc and rxc=rxc. Other clocking options can be used but line errors may occur due to the loopback=echo implementation.
nrzi	no	Set the port to operate with **NRZ** data encoding. **NRZ** encoding maintains a constant voltage level when data is present (1) and does not not return to a zero voltage (0) until data is absent. The data is decoded as an absolute value based on the voltage level (0 or 1).

	yes	Set the port to operate with **NRZI** data encoding. **NRZI** encoding does a voltage transition when data is absent (0) and no voltage transition (no return to zero) when data is present (1). Hence, the name non-return to zero inverted. The data is decoded using relational decoding.
txc	txc	Transmit clock source will be the **TxCI** signal.
	rxc	Transmit clock source will be the **RxC** signal.
	baud	Transmit clock source will be the internal **baud rate generator**.
	pll	Transmit clock source will be the output of the **DPLL** circuit. This can only be set with NRZI data encoding.
	-txc	Transmit clock source will be the inverted **TxCI** signal.
rxc	rxc	Receive clock source will be the **RxC** signal.
	txc	Receive clock source will be the **TxCI** signal. This can only be used with transmit clock option txc=txc.
	baud	Receive clock source will be the internal **baud rate generator**.
	pll	Receive clock source will be the output of the **DPLL** circuit. This can only be set with NRZI data encoding.
	-rxc	Receive clock source will be the inverted **RxC** signal.
txd	txd	Transmit data is not inverted.
	-txd	Transmit data is inverted.
rxd	rxd	Receive data is not inverted.
	-rxd	Receive data is inverted.
mode	fdx	HDLC Full Duplex mode (Default mode).
	ibm-fdx	IBM Full Duplex mode (SDLC).
	ibm-hdx	IBM Half Duplex mode (SDLC).
	ibm-mpt	IBM Multipoint mode (SDLC).
signal	yes	Notify application of modem signal (RTS and CTS) changes.
	no	Do not notify application of modem signal (RTS and CTS) changes.
mtu	*integer*	Set the maximum transmit unit to *integer* bytes with 2064 bytes maximum.
mru	*integer*	Set the maximum receive unit to *integer* bytes with 2064 bytes maximum.
speed	*integer*	Set the baud rate to *integer* bits per second with a minimum rate of 9600 bps and a maximum of 2048000 bps. Zero is also valid when txc is set to txc or -txc.

There are also several single-word options that set one or more paramaters at a time:

Keyword	Equivalent to Options:
external	txc=txc rxc=rxc loop=no
sender	txc=baud rxc=rxc loop=no
internal	txc=pll rxc=pll loop=no
stop	speed=0

EXAMPLES

The following command sets the first port to loop internally, use internal clocking and operate at 38400 baud:

example# hsip_init hihp0 38400 loop=yes
port=hihp0
speed=38400,
mode=fdx, signal=no, loopback=yes, nrzi=no, mtu=2064, mru=2064,
txc=baud, rxc=baud, txd=txd, rxd=rxd

The following command sets the same port's clocking, local loopback and baud rate settings to their default values:

example# hsip_init hihp0 speed=1536000 loopback=no txc=txc rxc=rxc
port=hihp0
speed=1536000,
mode=fdx, signal=no, loopback=no, nrzi=no, mtu=2064, mru=2064,
txc=txc, rxc=rxc, txd=txd, rxd=rxd

SEE ALSO

hsip_loop(1M), hsip_stat(1M), Intro(2), hsip(7D)

DIAGNOSTICS

device **missing minor device number**
The name *device* does not end in a decimal number that can be used as a minor device number.

bad speed: *arg*
The string *arg* that accompanied the "speed=" option could not be interpreted as a decimal integer.

Bad arg: *arg*
The string *arg* did not make sense as an option.

ioctl failure code = *errno*
An **ioctl(2)** system called failed. The meaning of the value of *errno* may be found in the **Intro(2)** manual page.

WARNINGS

hsip_init should not be used on an active serial link, unless needed to resolve an error condition. It should not be run casually, or if the user is unsure of the consequences of its use.

NAME

hsip_loop − high speed synchronous serial loopback test program for high speed serial interface.

SYNOPSIS

/opt/SUNWconn/bin/hsip_loop [−cdlsvt] *device*

DESCRIPTION

The hsip_loop command performs several loopback tests that are useful in exercising the various components of a serial communications link.

Before running a test, hsip_loop opens the designated port and configures it according to command line options and the specified test type. It announces the names of the devices being used to control the hardware channel, the channel number (ppa) corresponding to the *device* argument, and the parameters it has set for that channel. It then runs the loopback test in three phases.

The first phase is to listen on the port for any activity. If no activity is seen for at least four seconds, hsip_loop proceeds to the next phase. Otherwise, the user is informed that the line is active and that the test cannot proceed, and the program exits.

In the second phase, called the "first-packet" phase, hsip_loop attempts to send and receive one packet. The program will wait for up to four seconds for the returned packet. If no packets are seen after five attempts, the test fails with an error message. If a packet is returned, the result is compared with the original. If the length and content do not match exactly, the test fails.

The final phase, known as the "multiple-packet" phase, attempts to send many packets through the loop. Because the program has verified the integrity of the link in the first-packet phase, the test will not fail after a particular number of timeouts. If a packet is not seen after four seconds, a message is displayed. Otherwise, a count of the number of packets received is updated on the display once per second. If it becomes obvious that the test is not receiving packets during this phase, the user may wish to stop the program manually. The number and size of the packets sent during this phase is determined by default values, or by command line options. Each returned packet is compared with its original for length and content. If a mismatch is detected, the test fails. The test completes when the required number of packets have been sent, regardless of errors.

After the multiple-packet phase has completed, the program displays a summary of the hardware event statistics for the channel that was tested. The display takes the following form:

Port	**CRC errors**	**Aborts**	**Overruns**	**Underruns**	In	<-**Drops**->	**Out**
hihp0	**0**	**0**	**0**	**0**	**0**		**0**

This is followed by an estimated line speed, which is an approximation of the bit rate of the line, based on the number of bytes sent and the actual time that it took to send them. This is a very rough approximation and should not be used in bechmarking, because elapsed time includes time to print to the display.

OPTIONS

The options for hsip_loop are described in the following table:

Option	Parameter	Default	Description
−c	*packet_count*	100	Specifies the number of packets to be sent in the multiple-packet phase.
−d	*hex_data_byte*	*random*	Specifies that each packet will be filled with bytes with the value of *hex_data_byte*.
−l	*packet_length*	100	Specifies the length of each packet in bytes with a maximum of 2064 bytes.
−s	*line_speed*	9600	Bit rate in bits per second, minimum of 9600 bps and a maximum of 2048000 bps.
−v			Sets verbose mode. If data errors occur, the expected and received data is displayed.
−t	*test_type*	*none*	A number, from 1 to 4, that specifies which test to perform. The values for *test_type* are as follows:

1 Internal loopback test. Port loopback is on. Transmit and receive clock sources are internal (baud rate generator).

2 External loopback test. Port loopback is off. Transmit and receive clock sources are internal. Requires a loopback plug suitable to the port under test.

3 External loopback test. Port loopback is off. Transmit and receive clock sources are external (modem). Requires that one of the local modem or the remote modem be set in a loopback configuration.

4 Test using predefined parameters. User defines hardware configuration and may select port parameters using the **hsip_init(1M)** command.

All numeric options except **−d** are entered as decimal numbers (for example, **−s 19200**). If you do not provide the **−t** *test_type* option, hsip_loop prompts for it.

EXAMPLES

The following command causes hsip_loop to use a packet length of 512 bytes over the first CPU port:

example# hsip_loop −l 512 hihp0

In response to the above command, hsip_loop prompts you for the test option you want.

The following command performs an internal loopback test on the first CPU port, using 5000 packets and a bit rate of 56000 bps :

example# hsip_loop −t 1 −s 56000 −c 5000 hihp0

SEE ALSO

hsip_init(1M), hsip_stat(1M), hsip(7D)

DIAGNOSTICS

device **missing minor device number**
The name *device* does not end in a decimal number that can be used as a minor device number.

invalid packet length: *nnn*
The packet length was specified to be less than zero or greater than 2064.

poll: nothing to read
poll: nothing to read or write.
The **poll(2)** system call indicates that there is no input pending and/or that output would be blocked if attempted.

len *xxx* **should be** *yyy*
The packet that was sent had a length of *yyy*, but was received with a length of *xxx*.

nnn **packets lost in outbound queueing**
nnn **packets lost in inbound queueing**
A discrepancy has been found between the number of packets sent by hsip_loop and the number of packets the driver counted as transmitted, or between the number counted as received and the number read by the program.

WARNINGS

To allow its tests to run properly, as well as prevent disturbance of normal operations, hsip_loop should only be run on a port that is not being used for any other purpose at that time.

NAME

hsip_stat − report driver statistics from a high speed synchronous serial link port.

SYNOPSIS

/opt/SUNWconn/bin/hsip_stat [-f] -a|*num_of_ports*
/opt/SUNWconn/bin/hsip_stat [-f] *device* **[***period***]**
/opt/SUNWconn/bin/hsip_stat -c [-f] -a|*num_of_ports*
/opt/SUNWconn/bin/hsip_stat -c [-f] *device*

DESCRIPTION

The hsip_stat command reports the event statistics maintained by a high speed synchronous serial device driver. The report may be a single snapshot of the accumulated totals, or a series of samples showing incremental changes.

Event statistics are maintained by a driver for each physical channel that it supports. They are initialized to zero at the time the driver module is loaded into the system when one of the driver's entry points is first called.

The **device** argument is the name of the high speed serial device as it appears in the /dev directory. For example, **hihp0** specifies the first on-board high speed serial device.

As an alternative, you can display or clear the statistics for multiple physical channels using **num_of_ports** argument. The hsip_stat program will then display statistics accumulated for the first **n** number of ports, where **n** is **num_of_ports**.

The following is a breakdown of hsip_stat output:

speed	The line speed the device has been set to operate at. It is the user's responsibility to make this value correspond to the modem clocking speed when clocking is provided by the modem.
ipkts	The total number of input packets.
opkts	The total number of output packets.
undrun	The number of transmitter underrun errors.
ovrrun	The number of receiver overrun errors.
abort	The number of aborted received frames.
crc	The number of received frames with CRC errors.
isize	The average size (in bytes) of input packets.
osize	The average size (in bytes) of output packets.
ierror	Input error count (errors: Incomplete Frame, Empty frame, Glitch on RxC).
oerror	Output error count (errors: CTS lost, Glitch on TxC).
iutil	Input line utilization expressed as a percentage.
outil	Output line utilization expressed as a percentage.

OPTIONS

−f Select a complete set of accumulated statistics for the device specified. This is useful while debugging the **hsip** driver.

−a Select all devices.

−c Clear the accumulated statistics for the device specified. This may be useful when it is not desirable to unload a particular driver, or when the driver is not capable of being unloaded.

num_of_ports
Specify the number of devices that you want to dump the statistics.

period Cause hsip_stat to sample the statistics every *period* seconds and report incremental changes. The output reports line utilization for input and output in place of average packet sizes. These are the relationships between bytes transferred and the speed, expressed as percentages. The loop repeats indefinitely, with a column heading printed every twenty lines for convenience.

EXAMPLES

example# **hsip_stat hihp0**

speed	ipkts	opkts	undrun	ovrrun	abort	crc	isize	osize
9600	15716	17121	0	0	1	3	98	89

example# **hsip_stat 5**

	speed	ipkts	opkts	undrun	ovrrun	abort	crc	isize	osize
hihp0	9600	15716	10100	0	0	1	3	98	89
hihp1	9600	15234	20100	0	0	1	3	98	89
hihp2	9600	15123	18254	0	0	1	3	98	89
hihp3	9600	15378	18234	0	0	1	3	98	89
hihp4	9600	13900	13000	0	0	1	3	98	89

example# **hsip_stat -c hihp0**

speed	ipkts	opkts	undrun	ovrrun	abort	crc	isize	osize
9600	0	0	0	0	0	0	0	0

example# **hsip_stat hihp0 5**

ipkts	opkts	undrun	ovrrun	abort	crc	iutil	outil
12	10	0	0	0	0	5%	4%
22	60	0	0	0	0	3%	90%
36	14	0	0	0	1	51%	2%

(In this final example a new line of output is generated every five seconds.)

SEE ALSO

hsip_init(1M), hsip_loop(1M), hsip(7D)

DIAGNOSTICS

bad interval: *arg*
The argument *arg* is expected to be an interval and could not be understood.

device **missing minor device number**

The name *device* does not end in a decimal number that can be used as a minor device number.

WARNINGS

Underrun, overrun, frame-abort and CRC errors have a variety of causes. Communication protocols are typically able to handle such errors and initiate recovery of the transmission in which the error occurred. Small numbers of such errors are not a significant problem for most protocols. However, because the overhead involved in recovering from a link error can be much greater than that of normal operation, high error rates can greatly degrade overall link throughput. High error rates are often caused by problems in the link hardware, such as cables, connectors, interface electronics or telephone lines. They may also be related to excessive load on the link or the supporting system.

The percentages for input and output line utilization reported when using the *interval* option may occasionally be reported as slightly greater than 100% because of inexact sampling times and differences in the accuracy between the system clock and the modem clock. If the percentage of use greatly exceeds 100%, or never exceeds 50%, then the baud rate set for the device probably does not reflect the speed of the modem.

NAME

rscadm − administer SUN(tm) Remote System Control (RSC)

SYNOPSIS

rscadm help
rscadm resetrsc [-s]
rscadm set *variable value*
rscadm download [boot] *file*
rscadm show [variable]
rscadm date [-s] | [[mmdd]HHMM | mmddHHMM[cc]yy][.SS]
rscadm send_event [-c] *message*
rscadm modem_setup
rscadm useradd *username*
rscadm userdel *username*
rscadm usershow [username]
rscadm userpassword *username*
rscadm userperm *username* [cuar]

DESCRIPTION

rscadm administers the SUN(tm) Remote System Console (RSC). It allows the host server to interact with the RSC. The following operations are supported:

rscadm help
: Displays a usage screen.

rscadm resetrsc
: Reset the RSC. There are two types of reset allowed, a "hard" reset and a "soft" reset. The hard reset is done by default. The soft reset can be selected by using the -s option.

rscadm set
: Set RSC configuration variables. Examples of RSC configuration variables include RSC IP address and RSC hostname. See the RSC documentation for a complete list of RSC configuration variables.

rscadm download
: Program the RSC's firmware. There are two parts to the firmware, the boot monitor and the main image. By default, rscadm download programs the main firmware image. The boot option selects programming of the boot monitor.

rscadm show
: View the current RSC configuration variable settings. If no variable is specified, rscadm shows all variable settings.

rscadm date
: Show or set RSC's time and date. The -s options can be used to set RSC's time and date to the hosts time and date.

rscadm send_event
: Send a text based event to RSC. RSC may forward the event based on its event configuration.

rscadm modem_setup
: Direct connection to the RSC modem. This allows the user to enter AT commands to configure the modem. "~." returns to prompt.

rscadm useradd
: Add user account to RSC. RSC can support up to four separate users.

rscadm userdel
: Delete a user account from RSC.

rscadm usershow
: Show details on the specified user account. If a username is not specified, all user accounts will be shown.

rscadm userpassword
: Set a password for the user account specified. This password overrides any existing password currently set. There is no verification of the old password before setting the new password. See the RSC documentation on valid password formats.

rscadm userperm
: Set the authorization profile for the user. See the userperm options section in this man page for more detail.

OPTIONS

The following options are supported for rscadm:

rscadm resetrsc

[-s] Perform a "soft" reset instead of a "hard" reset. A hard reset physically resets the RSC hardware. The RSC software jumps to the boot firmware, simulating a reset, for a soft reset.

rscadm download

[boot] Program the boot monitor portion of the flash. The main portion of the flash is usually programmed.

rscadm show

[variable] Show the value of that particular variable.

rscadm date

[-s] Set the date to the hosts time and date.

[[mmdd]HHMM | mmddHHMM[cc]yy][.SS]
: the date.

mm - month
dd - day
HH - hour
MM - minute
cc - the first two digits of the four digit year
yy - last 2 digits of the year number
SS - seconds

rscadm send_event

[-c] Send a critical event. Without the -c, send_event sends a warnings. Warnings are only logged in the RSC event log and not forwarded further.

rscadm usershow

[username] RSC account name to display info on. If no username is given, all accounts will be displayed.

rscadm userperm

[cuar] Set permissions for RSC account. If no permissions are specified, all four permissions will be disabled. The options are to; allow user to connect to (c)onsole, allow user to use the (u)ser commands to modify RSC accounts, allow user to (a)dminister/change the RSC configuration variables, allow the user to (r)eset RSC and to power on/off the host.

OPERANDS

The following operands are supported for rscadm:

rscadm set

variable RSC configuration variable to set. See the RSC documentation for a list of configuration variables.

value Value to set RSC configuration variable to. See the RSC documentation for a list of valid values.

rscadm download

file Firmware file to download. The file should contain the RSC boot monitor image or RSC main image.

rscadm send_event

message Text message to describe event. Should be enclosed in quotes.

rscadm useradd

username Username for new RSC account.

rscadm userdel

username RSC account to be removed.

rscadm userpassword

username RSC account to have password set.

rscadm userperm

username RSC account to have permissions changed.

EXIT STATUS

= 0 on success
!= 0 on failure (with status message)

EXAMPLES

```
# rscadm date
# rscadm date -s
# rscadm date 050113101998
```

```
# rscadm set hostname rsc15
# rscadm show
# rscadm show hostname
# rscadm send_event -c "The UPS signaled a loss in power!"
# rscadm send_event "The disk is close to full capacity"
# rscadm useradd rscroot
# rscadm userdel olduser
# rscadm usershow
# rscadm usershow rscroot
# rscadm userperm rscroot cuar
# rscadm userperm newuser c
# rscadm userperm newuser
```

NOTES

rscadm modem_setup - "~." will only work after a new line.

rscadm MUST be run as root.

BUGS

None known.

NAME

sunvts − Invokes the SunVTS kernel and its user interface

SYNOPSIS

sunvts [−**lepqstv**] [−**o** *option_file*] [−**f** *log_dir*] [−**h** *hostname*]

AVAILABILITY

SUNWvts

DESCRIPTION

The **sunvts** command is used to invoke the SunVTS user interface and kernel on the same system. It could be used to start the user interface on the local system and connect to the SunVTS kernel on the remote system. By default, it displays CDE Motif graphic interface for CDE environment, OpenLook graphic interface for OpenWindows environment, or TTY interface for non-windowing system.

OPTIONS

−**l** Displays SunVTS OpenLook graphic interface.

−**e** Disables the security checking feature.

−**f** *log_dir*
Specifies an alternative log_file directory. The default log_file directory is **/var/opt/SUNWvts/logs.**

−**h** *hostname*
Starts the SunVTS user interface on the local system, which connects to or invokes the SunVTS kernel on the specified host after security checking succeeds.

−**o** *option_file*
Starts the SunVTS kernel with the test options loaded from the specified *option_file,* which by default is located in **/var/opt/SUNWvts/options.**

−**p** Starts the SunVTS kernel **vtsk (1M)** such that it does not probe the test system's devices.

−**q** Automatically quits both the SunVTS kernel and the user interface when testing stops.

−**s** Automatically starts testing from a selected group of tests. The flag must be used with the −**o** *option_file* flag.

−**t** Starts **vtstty (1M),** a TTY based interface, instead of CDE or OpenLook interface.

−**v** Displays version information from **vtsui**(1M) and **vtsk**(1M).

NOTES

If **vtsk (1M)** is already running on the test system, the **sunvts** command ignores the −**e,** −**o,** −**f,** −**q,** −**p,** and −**s** options.

SEE ALSO

vtsk(1M), **vtstty**(1M), **vtsui**(1M), **vtsui.ol**(1M), **vtsprobe**(1M)

NAME

vts_cmd − Send a command to the SunVTS kernel (vtsk)

SYNOPSIS

vts_cmd [*command*] [*argument*]

AVAILABILITY

SUNWvts

DESCRIPTION

vts_cmd is a UNIX shell application that allows you to send a single command to the SunVTS kernel (vtsk). The test machine's SunVTS kernel will send the response to the standard output.

The SunVTS application programming interface (API) is character based, which means that a string of characters (in the form of a *command*) can be sent to the SunVTS kernel, which then returns a reply back in the form of a string of characters.

vts_cmd(1M) allows the user to send commands and receive replies from a UNIX command line.

OPTIONS

vts_cmd uses the commands listed below. In all cases, the commands (and any of the command's arguments) must follow vts_cmd. See the EXAMPLES section for reference.

Some of the command descriptions listed below refer to a testnode. In the SunVTS API, there is a hierarchy of testnodes, with the system being on the top, the test groups below the system, and the tests themselves at the bottom. In the commands below, use a slash "/" to refer to the system. A test group can be one of the following: Processor(s), Memory, Network, SCSI-Devices(esp0), Comm.Ports, Graphics, OtherDevices, or any user specified group. When referring to a test, you must mention the device name and the test name [for example, sound0(audio)].

list testnode

Displays all the testnodes under the specified testnode.

config testnode

Displays the configuration information of the testnode.

status [testnode] [-r]

Displays the testing status information of the system. If a testnode is specified, status will display the status information of that testnode. If you use the -r argument, the status information of all of testnodes recursive to the testnode will be displayed.

option [testnode] [-l] [-h|n|s|t|a]

Either displays all the options associated with the specified testnode, or sets a specific option in a testnode.

To display a testnode's options, type option followed by the testnode and one of the categories:

-h **Threshold**
-n **Notify category**
-s **Scheduling category**

-t Test execution category
-a Advanced category

vts_cmd will print all options, as well as the setting of each option. Use the -l option to display the options in long form. In long form, the options will be displayed with all their settings.

option [testnode] [test_option] [-g|s|x|y|z]

-g is used to pass all of the current option settings, for a given instance of a given test, to all of the same instances and tests that are in the same group (will not affect the same tests that are in different groups).

-s is used to pass all of the current option settings for a given instance of a given test, to all of the same instances for all of the same tests on the system (rather than for a group, as with -g).

-x is used to pass all of the current option settings for a given instance of a given test, to all the instances of that test.

-y is used to pass all of the current option settings for a given instance of a given test, to all the instances of all the same tests in a particular group.

-z is used to pass all of the current option settings for a given instance of a given test, to all the instances of all the same tests in the whole system.

To set an option, you must state the testnode immediately followed by the option and the new setting. You must use this format when setting an option:

vts_cmd option testnode[option:setting]

Once the option has been successfully changed, vts_cmd will display the word "DONE".

select testnode

Selects a testnode. If a testnode is selected, all the tests associated with the testnode will be enabled and run when testing begins.

For example, if you select the Graphics testnode, all the tests in Graphics will be enabled for testing. If you select just the "fpu(fputest)" test, then you will only enable this test.

deselect testnode

Deselects a testnode. If a testnode is deselected, all the tests associated with the testnode will be disabled and will not be run when testing begins.

For example, if you deselect the OtherDevices testnode, all the tests in the OtherDevices will be disabled. If you select just the "cgsix0(cg6)" test, then you will only enable this test.

start

Starts all enabled (selected) SunVTS tests.

stop

Stops all running SunVTS tests.

suspend

Suspends (or pauses) all running SunVTS tests. When you are ready to resume testing, type "resume".

resume

Resumes any suspended tests.

reset

Resets all the SunVTS pass and error counts to zero.

probe

Probes all the devices on the test machine and updates the SunVTS kernel's device list.

If a device is listed in the device list, but it is not found during the probe, it will be removed from the list. Conversely, if a device does not exist in a previous device list and is found during the probe, it will be added to the list.

load option_file

Loads an option file. Once loaded, the system and test options will be changed to reflect the settings listed in the option file.

Option files are stored in the /var/opt/SUNWvts/options directory.

store option_file

Creates an option file, listing all the system and test options, and save it in the /var/opt/SUNWvts/options directory.

quit
Terminates the SunVTS kernel (vtsk).

invokeds
Starts the deterministic scheduler.

quitds
Terminates the deterministic scheduler.

loadseq sequence_file
Loads a sequence file. Once loaded, the deterministic scheduler UI will reflect the tasks in the loaded sequence file.

storeseq sequence_file
Creates sequence_file, listing all the tasks in the directory /var/opt/SUNWvts/sequences.

statusseq
Returns a string containing the status information of the currently running sequence. The string consists of four fields separated by commas (","). The fields are: current status of SunVTS, current loop count of the sequence, total loop count of the sequence, and currently running task's position.

startseq
Starts the execution of the deterministic scheduler.

stopseq
Stops the execution of the currently running task in the sequence file. Upon starting again, the execution will start from the tast that was stopped.

resumeseq
Restarts the execution of the sequence file. Execution will start at the point where the sequence was stopped, unless the sequence was reset, in which case it would start at the beginning of the sequence file.

resetseq
Sets the starting point of the execution to the start of the sequence file. Will also reset the passes and error count.

suspendseq
Suspends the execution of the currently running task in the sequence file.

removeseq sequence_file
Removes sequence_file from the list of sequence files in the directory /var/opt/SUNWvts/sequences.

listtask
Lists the tasks that are present in the currently loaded

sequence file.

addtask task_name [i]
Adds task_name at the ith position in the sequence file. If no index is passed then the task would be added to the end of the list.

deletetask [i]
Removes the task at the specified index from the selected sequence.

loadtask task_name
Loads a task file. Once loaded, the system and test options will be changed to reflect the settings listed in the task file.

setloopcount count
Sets the number of loops to run in the current sequence to count.

getvtsmode
Gets the current mode of SunVTS kernel.

EXAMPLES

To list out the configuration information of the test machine, you would use the config command:

```
sample% vts_cmd config /
/[Hostname:sample,Model:SPARCstation-10,SunVTS version:1.0]:idle
```

To load an option file, you would use the load command:

```
sample% ls /var/adm/sunvtslog/options
CPU_options     sample          options
sbus_standard
sample% vts_cmd load sbus_standard
DONE
```

To print all the system options in the Comm.Ports testnode, you would use the option command and pipe the output to your local printer:

```
sample% vts_cmd option Comm.Ports -l | lp
request id is printer-213 (standard input)
```

ENVIRONMENT

VTS_CMD_HOST=hostname
The hostname of the test machine running the SunVTS kernel **(vtsk).** If this environment variable is not set, **vts_cmd** will attempt to send the commands to

the local machine's SunVTS kernel.

SEE ALSO SunVTS User's Guide

NAME

vtsk − SunVTS diagnostic kernel

SYNOPSIS

vtsk [−**epqsv**] [−**o** *options_file*] [−**f** *logfile_directory*]

AVAILABILITY

SUNWvts

DESCRIPTION

The **vtsk** command starts up the SunVTS diagnostic kernel as a background process. There can only be one copy of **vtsk** running at a time. Only the superuser can execute this command.

Normally, **vtsk** is automatically started up by the **sunvts (1M)** command if it is not already running. **vtsk** will also be invoked by **inetd (1M)** when there is a connection request from vtsui or vtsui.ol. In that case, the security file, **.sunvts_sec,** will be checked for the permission before running vtsk on the target host specified by **vtsui**(1M) or **vtsui.ol**(1M).

OPTIONS

−**e** Enables the security checking for all connection requests.

−**p** Starts SunVTS diagnostic kernel, but does not probe system configuration.

−**q** Quits both the SunVTS diagnostic kernel and the attached User Interfaces when the testing is completed.

−**s** Runs enabled tests immediately after started.

−**v** Display SunVTS diagnostic kernel's version information only.

−**o** *options_file*
Starts the SunVTS diagnostic kernel and sets the test options according to the option file named *options_file.*

−**f** *logfile_directory*
Specifies an alternative logfile directory, other than the default.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

−**1** An error occurred.

FILES

/var/opt/SUNWvts/options default option file directory.
/var/opt/SUNWvts/logs default log file directory.

SEE ALSO

sunvts(1M), **vtsui**(1M), **vtsui.ol**(1M), **vtstty**(1M), **vtsprobe**(1M)

NAME

vtsprobe − prints the device probe information from the SunVTS kernel

SYNOPSIS

vtsprobe [−**m**] [−**h** *hostname*]

AVAILABILITY

SUNWvts

DESCRIPTION

vtsprobe is a utility that displays the device and configuration information contained in the SunVTS kernel. The output includes the SunVTS assigned group for the device, the device name, the device instance, the testname attached to this device, and the configuration information obtained from the device-specific test probe.

OPTIONS

−**m** Specifies manufacturing mode, which displays the probe information in a format that is easy to read using script files.

−**h** *hostname*
Specifies the *hostname* to connect to and get the device and configuration information. If not specified, the current host will be used.

USAGE

After the SunVTS kernel is up and running, you may type **vtsprobe** at the shell prompt to get the probe output. (See the **sunvts (1M)** man page for more information on how to start up SunVTS.

EXAMPLE

Running **vtsprobe** on a sun4m SPARCclassic produces the following output:

```
% vtsprobe

Processor(s)
        system(systest)
                System Configuration=sun4m SPARCclassic
                System clock frequency=50 MHz
                SBUS clock frequency=25 MHz
        fpu(fputest)
                Architecture=sparc
                Type=TI TMS390S10 or TMS390S15 microSPARC chip
Memory
        kmem(vmem)
                Total: 143120KB
        mem(pmem)
                Physical Memory size=24 Mb
SCSI-Devices(esp0)
        c0t2d0(rawtest)
                Capacity: 638.35MB
                Controller: esp0
                Vendor: MICROP
                SUN Id: 1588-15MBSUN0669
                Firmware Rev: SN0C
```

```
            Serial Number: 1588-15MB103
        c0t2d0(fstest)
            Controller: esp0
        c0t3d0(rawtest)
            Capacity: 404.65MB
            Controller: esp0
            Vendor: SEAGATE
            SUN Id: ST1480   SUN0424
            Firmware Rev: 8628
            Serial Number: 00836508
        c0t3d0(fstest)
            Capacity: 404.65MB
            Controller: esp0
            Vendor: SEAGATE
            SUN Id: ST1480   SUN0424
            Firmware Rev: 8628
            Serial Number: 00836508
        c0t3d0(fstest)
            Controller: esp0
        c0t6d0(cdtest)
            Controller: esp0
        tape1(tapetest)
            Drive Type: Exabyte EXB-8500 8mm Helical Scan
Network
        isdn0(isdntest)
            NT Port  TE Port
        le0(nettest)
            Host_Name: ctech84
            Host Address: 129.146.210.84
            Host ID: 8001784b
            Domain Name: scsict.Eng.Sun.COM
Comm.Ports
        zs0(sptest)
            Port a -- zs0  /dev/term/a : /devices/ ... a
            Port b -- zs1  /dev/term/b : /devices/ ... b
Graphics
        cgthree0(fbtest)

OtherDevices
        bpp0(bpptest)
            Logical name: bpp0
        sound0(audio)
            Audio Device Type: AMD79C30
        sound1(audio)
            Audio Device Type: DBRI Speakerbox
```

spd0(spdtest)
Logical name: spd0

NOTES

The output of **vtsprobe** is highly dependent on the device being correctly configured into the system (so that a SunVTS probe for the device can be run successfully on it) and on the availability of a device-specific test probe.

If the device is improperly configured or if there is no probing function associated with this device, **vtsprobe** cannot print any information associated with it.

SEE ALSO

sunvts(1M), **vtsk**(1M), **vtsui**(1M), **vtsui.ol**(1M), **vtstty**(1M)

NAME

vtstty − TTY interface for SunVTS

SYNOPSIS

vtstty [−**qv**] [−**h** *hostname*]

AVAILABILITY

SUNWvts

DESCRIPTION

vtstty is the default interface for SunVTS in the absence of a windowing environment. It can be used in a non-windowing environment such as a terminal connected to the serial port of the system. However, its use is not restricted to this; **vtstty** can also be used from shell window.

OPTIONS

−**q** The "auto-quit" option automatically quits when the conditions for SunVTS to quit are met.

−**v** Prints the **vtstty** version. The interface is not started when you include this option.

−**h** *hostname*
Connects to the SunVTS kernel running on the host identified by *hostname.*

USAGE

The **vtstty** screen consists of four panels: main control, status, test groups, and console. The panels are used to display choices that the user can select to perform some function and/or to display information. A panel is said to be "in focus" or in a "selected" state when it is surrounded by asterisks and the current item is highlighted. In order to choose from the items in a panel, the focus should be shifted to that panel first.

The following are the different types of selection items that can be present in a panel:

Text string	Describes a choice that, when selected, either pops up another panel or performs a function. For example, "stop" will stop the SunVTS testing.
Data entry field	To enter or edit numeric or textual data.
Checkbox	Represented as "[ ]". Checkboxes are associated with items and indicate whether the associated item is selected or not. A checkbox can be in one of the following two states: Deselected [ ] or Selected [*].

The key assignments given below describe the keys for shifting focus, making a selection, and performing other functions:

TAB or <CTRL>W	Shift focus to another panel
RETURN	Select current item
Spacebar	Toggle checkbox
Up arrow or <CTRL>U	Move up one item
Down arrow or <CTRL>N	Move down one item

Left arrow or <CTRL>P	Move left one item
Right arrow or <CTRL>R	Move right one item
Backspace	Delete text in a data entry field
ESC	Dismiss a pop-up
<CTRL>F	Scroll forward in a scrollable panel
<CTRL>B	Scroll backward in a scrollable panel
<CTRL>X	Quit **vtstty** but leave the SunVTS kernel running
<CTRL>L	Refresh the **vtstty** screen

NOTES

1. To run **vtstty** from a telnet session, carry out the following steps:

   a. Before telnet-ing, determine the values for "rows and "columns". (See **stty**(1)).

   b. Set term to the appropriate type after telnet-ing(for example, **set term=vt100**

   c. Set the values of columns and rows to the value noted above. (See **stty**(1)).

2. Before running **vtstty** ensure that the environment variable describing the terminal type is set correctly.

SEE ALSO

sunvts(1M), **vtsk**(1M), **vtsui**(1M), **vtsui.ol**(1M), **vtsprobe**(1M)

NAME

vtsui − SunVTS Graphic User Interface (CDE)

SYNOPSIS

vtsui [−**qv**] [−**h** *hostname*]

AVAILABILITY

SUNWvts

DESCRIPTION

The **vtsui** command starts up the CDE Motif version of SunVTS graphic user interface. There can be multiple instances of **vtsui** running at the same time, all connected to one SunVTS diagnostic kernel, **vtsk**(1M). The name of the host machine running the diagnostic kernel, **vtsk**(1M), will be displayed in the title bar of the graphical user interface window.

vtsui is automatically started up by the **sunvts (1M)** command. **vtsui can** be also used to start **vtsk (1M)** if **inetd (1M)** is in operation. In that case, the security file, **sunvts_sec,** will be checked for the permission before running **vtsk** on the target host.

See the "SunVTS User's Guide" for a complete description on using the graphical user interface.

OPTIONS

−**q** Quits the SunVTS graphic user interface when testing has terminated.

−**v** Displays graphic user interface version information only.

−**h** *hostname*
Starts the SunVTS graphic user interface and connects to the SunVTS diagnostic kernel running on *hostname,* or invokes the kernel if not running, after security checking succeeds. If *hostname* not specified, the local host is assumed.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

1 An error occurred.

SEE ALSO

sunvts(1M), **vtsk**(1M), **vtsui.ol**(1M), **vtstty**(1M), **vtsprobe**(1M)

NAME

envmond.conf - configuration file for environment monitor daemon

SYNOPSIS

/usr/platform/SUNW,UltraSPARC-IIi-Netract/envmond.conf

DESCRIPTION

The **envmond.conf** file is the configuration file for **envmond**(1M), the system environment monitor daemon. The daemon monitors environmental devices to check for conditions that may require some action. The **envmond (1M)** daemon logs appropriate messages to a system log file via **syslogd**(1M).

Each configuration file entry provides the daemon information about a shared object library, referred to as a policy, which has the knowledge to monitor a device. Each policy entry describes an interface between the envmond daemon and the policy. The policy entry in the **envmond.conf** file can contain configurable parameters in the *policy-args* field.

All policy entries have the same format:

poll-interval policy-name policy-args

The three fields shown above are separated by whitespace. Use the backslash (/) at the end of a line to continue *policy-args* to the line following.

The fields in the **envmond.conf** file are described as follows:

poll-interval

Given in seconds as a decimal number, specifies how often to invoke the policy check function. If *poll-interval* is 0, the policy check function will never be called.

policy-name

The file name, with optional path, of the file implementing the policy. The default location for the policy files is **/usr/platform/SUNW,UltraSPARC-IIi-Netract/lib/envmond/sparcv9**

policy-args

An optional list of whitespace-separated arguments to be passed to the policy during initialization. The number and format of these arguments is policy-dependent.

The following sections describe policies shipped with the implementation of **envmond***(1M)*.

fancpu Policy

The fancpu policy polls I2C slave devices every *poll-interval* seconds to get the current CPU temperature and the fantray status. If the CPU temperature reaches a warning temperature threshold, a warning message is printed on the system console and to the system log file specified in **syslog.conf**(4). If the CPU temperature reaches the shutdown temperature, a critical error message is printed on the system console by **syslogd**(1M). The system is then halted by the **shutdown**(1M) command. The fan status will be reflected by the corresponding LEDs on the System Status Board, and with log messages sent to **syslogd.**

powersupply Policy

The powersupply policy sets and clears the power supply LEDs on the System Status Board to reflect power supply status. The policy also handles an interrupt event if a power supply fails.

scsb Policy

The System Controller and Status Board Policy is primarily to configure the scsb driver for cPCI Slot Status LED control. The default **scsb_led_ctrl** setting is false, meaning that the scsb driver controls the cPCI slot LEDs. If **scsb_led_ctrl** is set to true, then some application is responsible for slot LED updates.

EXAMPLES

Example 1: Sample Entries

The first entry, below, invokes the powersupply shared library every 60 seconds. The second entry specifies that the scsb policy controls the cPCI Slot Status LED.

```
60 powersupply.so
scsb.so scsb_led_crtl=false
```

FILES

/usr/platform/SUNW,UltraSPARC-IIi-Netract/
Installation directory.

The following relative pathnames are all beneath the directory named above.

lib/envmond/sparcv9/envmond
Executable for the environmental daemon.

lib/envmond/sparcv9/fancpu.so
Policy for CPU temperature and fan speed control.

lib/envmond/sparcv9/powersupply.so
Policy for power supply monitoring.

SEE ALSO

envmond(1M), **syslogd**(1M), **syslogd.conf**(4)

NAME

hsip − PCI-Bus based high speed serial line interface.

SYNOPSIS

#include <fcntl.h>
#include </usr/include/sys/ser_sync.h>
open(/dev/hihp*n*, mode);
open(/dev/hihp, mode);

DESCRIPTION

The **hsip** module is a loadable and unloadable STREAMS driver that implements the sending and receiving of data packets such as HDLC frames over synchronous serial lines. The **hsip** driver is a standalone driver that supports HSI/P PCI-Bus based serial interface hardware and provides phsipcal level data transfer services for upper data link layer protocols (e.g. HDLC or SDLC).

The **hihp***n* devices provide what is known as a **data path** which supports the transfer of data via **read**(2) and **write**(2) system calls, as well as **ioctl**(2) calls. Data path opens are exclusive in order to protect against injection or diversion of data by another process.

The **hihp** 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.

The HSIP ports support several options for **clock sourcing** and data encoding. Both the transmit and receive clock sources can be set to be the external transmit clock (TxC), external receive clock (RxC), 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 **hsip_loop(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 **hsip** 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. 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 **hsip** 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 baudrate.

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 baudrate 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 **hsip** ioctl() commands:

```
struct scc_mode {
        char    sm_txclock;     /* transmit clock sources */
        char    sm_rxclock;     /* receive clock sources */
        char    sm_iflags;      /* data and clock inversion flags (non-zsh) */
        u_char  sm_config;      /* boolean configuration options */
        int     sm_baudrate;    /* real baud rate */
        int     sm_retval;      /* reason codes for ioctl failures */
};

struct sl_stats {
        int     ipack;          /* input packets */
        int     opack;          /* output packets */
        int     ichar;          /* input bytes */
        int     ochar;          /* output bytes */
        int     abort;          /* abort received */
        int     crc;            /* CRC error */
        int     cts;            /* CTS timeouts */
```

```
        int     dcd;            /* Carrier drops */
        int     overrun;        /* receive overrun */
        int     underrun;       /* transmit underrun */
        int     ierror;         /* input error */
        int     oerror;         /* output error */
        int     nobuffers;      /* receive side memory allocation failure */
};
```

ERRORS

An **open()** will fail if a STREAMS message block cannot be allocated, or:

ENXIO — The unit being opened does not exist.

EBUSY — The device is in use by another serial protocol.

An **ioctl()** will fail if:

EINVAL — An attempt was made to select an invalid clocking source.

EINVAL — The baud rate specified for use with the baud rate generator would translate to a null time constant in the SCC's registers.

FILES

/dev/hihp[0-n], /dev/hihp
Character-special devices.

/usr/include/sys/ser_sync.h
Header file specifying synchronous serial communication definitions.

SEE ALSO

hsip_init(1M), hsip_loop(1M), hsip_stat(1M),

Refer to the *Motorola MC68360 Quad Integrated Communications Controller Technical Manual* for details of the SCC's operation and capabilities.

DIAGNOSTICS

hihp data open failed, no memory, rq=*nnn*
hihp 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).

hihp_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.

hihp: 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.

hihp*n*: **invalid operation for clone dev.**
An inappropriate STREAMS message type was passed through a control path. Only M_IOCTL and M_PROTO message types are permitted.

hihp*n*: **not initialized, can't send message**
An M_DATA message was passed to the driver for a channel that had not been programmed at least once since the driver was loaded. The S_IOCSETMODE ioctl command performs the programming operation.

hihp*n*: transmit hung
The transmitter was not successfully restarted after the watchdog timer expired.

Index