Solaris 2.5.1 Reference Manual for SMCC-Specific Software

Sun Microsystems Computer Company
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No: 802-5328-10
Revision A, May 1996

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAMS(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

NAME

symon – bring up the Solstice SyMON system monitor console

SYNOPSIS

symon [–**configReaderRoot** *root-node*] [–**ctr** *root-node*]
[–**heartbeatInterval** *intervals*] [–**hi** *intervals*] [–**interval** *intervals*] [–**i** *intervals*]
[–**kernelRoot** *root-node*] [–**kr** *root-node*] [–**minWait** *seconds*] [–**mw** *seconds*]
[–**pruneTime** *minutes*] [–**pt** *minutes*] [–**tempPruneTime** *minutes*] [–**tpt** *minutes*]
[–**rpcRetry** *seconds*] [–**rr** *seconds*] [–**rpcTimeout** *microseconds*] [–**rt** *microseconds*]
[–**serverTimeout** *seconds*] [–**st** *seconds*] [–**target** *machine*] [–**t** *machine*]
[–**vtsui** *file*] [–**help**] [–**h**] [–**?**]

AVAILABILITY

SUNWsymon

DESCRIPTION

symon is the primary user interface to the SyMON system monitor. Invoking symon brings up the launcher window, from which the seven SyMON consoles are launched:

- Event Display
- Data Catalog
- Physical View
- Logfile View
- Logical View
- Process View
- Diagnostic Screen

For further details on the operation of **symon** please see the *Solstice SyMON User's Guide.*

OPTIONS

–**configReaderRoot** Set the root node for Config Reader hierarchy (default is system).

–**ctr** Set the root node for Config Reader hierarchy (default is system).

–**heartbeatInterval** Set the polling time for the heartbeat check for agents (default is 10 intervals).

–**hi** Set the polling time for the heartbeat check for agents (default is 10 intervals).

–**interval** Set the polling interval for agents (default is 10 intervals).

–**i** Set the polling interval for agents (default is 10 intervals).

–**kernelRoot** Set Kernel Reader root node name (default is KernelReader).

–**kr** Set Kernel Reader root node name (default is KernelReader).

–**minWait** Set a minimum wait time between polls/updates (default is 1 second).

–**mw** Set a minimum wait time between polls/updates (default is 1 second).

–**pruneTime** Time after which unchanged data (old processes) is pruned from the **sm_krd** (Kernel Reader) hierarchy (default is 120 minutes).

–pt Time after which unchanged data (old processes) is pruned from the **sm_krd** (Kernel Reader) hierarchy (default is 120 minutes).

–tempPruneTime Time after which unchanged Config Reader data (board temperature) will be pruned from **sm_configd** hierarchy (default is 1440 minutes).

–tpt Time after which unchanged Config Reader data (board temperature) will be pruned from **sm_configd** hierarchy (default is 1440 minutes).

–rpcRetry Time between RPC timeout and retry (default is 2 seconds).

–RR Time between RPC timeout and retry (default is 2 seconds).

–rpcTimeout Timeout value for RPC (default is 0 microseconds).

–rt Timeout value for RPC (default is 0 microseconds).

–serverTimeout Time before declaring a server is dead (default is 60 seconds).

–st Time before declaring a server is dead (default is 60 seconds).

–target System to be monitored.

–t System to be monitored.

–vtsui Name of SunVTS user interface binary (default is **vtsui**).

–help Listing of arguments.

–h Listing of arguments.

–? Listing of arguments.

ENVIRONMENT

TCL_LIBRARY Location of the TCL library.

XFILESEARCHPATH Location of the X Files.

DTAPPSEARCHPATH Location of the CDE X Defaults files.

DTDATABASESEARCHPATH Location of the CDE database files.

DTHELPSEARCHPATH Location of the CDE help files.

XMICONSEARCHPATH Location of the **symon** icons.

FILES

common.tcl Common TCL routines for the display.

cpu_utilization.tcl TCL routines to define the chart for CPU utilization.

disk_service_time.tcl TCL routines to define the chart for disk service time.

memory_usage.tcl TCL routines to define the chart for memory usage.

init.tcl	TCL routines to initialize **symon.**
queue_lengths.tcl	TCL routines to define the chart for queue lengths.
sysmeter.tcl	TCL routines to define the chart for System Meters.

SEE ALSO

sm_configd(1M), **sm_egd**(1M), **sm_krd**(1M), **sm_logscand**(1M), **sm_symond**(1M), **sm_symond.conf**(4)

NAME

contrast – Adjust system screen contrast

SYNOPSIS

/usr/openwin/bin/contrast [–**k**] [–**ud** [**step**]]

AVAILABILITY

SUNWpmow

DESCRIPTION

contrast (1M) is a binary that can be used to adjust the system screen contrast level.

OPTIONS

–u step Increase contrast by step.

–d step Decrease contrast by step.

–k This is the repeater mode for the binary. The binary expects the letter ’k’ on its STDIN and increases or decreases the contrast by one until STDIN is closed by the other side. This is the mode in which the binary interacts with the **speckeysd (1M)** daemon.

NOTES

The contrast level is adjusted by communicating with the Power Management driver, **pm (7)** , through ioctls provided by the Power Management Framework.

SEE ALSO

speckeysd(1M), **pm**(7)

NAME

dtpower – desk–top power manager, system and device power management tool

SYNOPSIS

dtpower [*generic–tool–arguments*] [–**sampleTime** *n* **|** –**st** *n*]
[–**warnTime1** *n* **|** –**wt1** *n*] [–**warnTime2** *n* **|** –**wt2** *n*]
[–**nobell**]

AVAILABILITY

SUNWpmow

DESCRIPTION

dtpower provides a graphical user interface (GUI) to the power management system (see **pm**(7)). It allows the user to configure certain power manageable devices to shutdown after a specified period of inactivity. Different hardware platforms support different devices. Most platforms allow power management of display(s). Some platforms allow power management of disk drives. The set of power configurations for all devices is called a power profile.

dtpower also displays the current autoshutdown settings (see **powerd**(1M)). If **dtpower** is run as root, these settings may be changed. These settings are not included in a power profile.

If a battery is present, **dtpower** monitors the battery level. If the system is running from the battery, **dtpower** displays low power warnings when the battery charge is running low. **dtpower** maintains two device power profiles – one for use on AC and one for use with the battery. This enables you to customize your device power settings, depending on your power source. **dtpower** switches profiles automatically when the machine's power supply changes. There may be a small delay (about 30 seconds) before **dtpower** notices a change in power source.

You must be console owner or root to run **dtpower.**

OPTIONS

generic–tool–arguments
dtpower accepts the generic tool arguments described in **xview**(7).

–sampleTime n
–st n
dtpower continually checks the battery capacity, if a battery is present. This option sets the period of this check. The default is 10 seconds.

–warnTime1 n
–wt1
–warnTime2 n
–wt2
dtpower displays two warnings of low battery power. These options set the time before battery exhaustion at which the warnings will occur. The default warning times are at 10 and 5 minutes. Note that **powerd**(1M) will shut the system down when the battery is exhausted.

–nobell
By default, whenever **dtpower** displays a warning dialog, it sounds a bell. This option disables the bell.

USAGE

dtpower operates via a set of pull-down menus, slider(s) and buttons in a control panel. From the control panel you may access one other panel, the autoshutdown panel.

The Control Panel

Menu Bar

File

Exit Exits the application. If you have pending changes, you will prompted to apply or discard them before exiting.

Help

Help

Displays an overview of the **dtpower** application.

Information

With Battery

The charge level of the battery is displayed. If the battery is the connected power source, then the estimated battery life is displayed below the charge gauge.

You can select which device power profile to edit using the toggle buttons above the slider(s). Note that the active profile is determined by your power source, not the toggle buttons.

Without battery

The power profile displayed is for an AC power supply. There is no access to the battery power profile.

Slider(s)

Screen This slider shows the amount of time the keyboard and mouse will be unused before the screen turns off. To change this time, move the slider and select apply. To turn the screen on, move the mouse or press a key.

Disk This slider shows the amount of time the disk will be idle before spinning down. This is not available on all platforms. The disk will automatically spin up the next time it is needed.

Buttons

Apply This applies any changes to your active power profile and saves all settings into *$HOME/.pmrc* so they are remembered the next time the application is started.

Reset to Standard

This resets the active power profile to its default values and saves these into *$HOME/.pmrc.*

Set Autoshutdown...

This brings up the autoshutdown panel.

The Autoshutdown Panel

As root

This panel allows you to view and edit the parameters governing autoshutdown. The first box adjusts the amount of time the console keyboard and mouse must be unused before the system will auto-shutdown.

The toggle buttons beneath determine the times when auto-shutdown is in effect at all.

OK This applies any changes made and saves them to **power.conf**(4) as the default settings.

Cancel Dismisses the window and discards any changes

Help Displays a brief overview text.

As a regular user

This panel allows you to view the current settings. Changes are not permitted.

OK Disabled

Cancel Dismisses window

Help Displays a brief overview text.

FILES

$HOME/.pmrc Per user customized power profile

/etc/power.conf System-wide power configuration profile

/usr/openwin/lib/app-defaults/Dtpower
Text messages file

SEE ALSO

cpr(7), **pm**(7), **power.conf**(4), **pmconfig**(1M), **powerd**(1M)

NAME

ffbconfig – configure the FFB Graphics Accelerator

SYNOPSIS

/usr/sbin/ffbconfig [**–dev** *device-filename*]
[**–res** *video-mode* [**now** | **try**] [**noconfirm** | **nocheck**]]
[**–file machine** | **system**]
[**–deflinear** **true** | **false**]
[**–defoverlay** **true** | **false**]
[**–linearorder** **first** | **last**]
[**–maxwids** *n*] [**–propt**] [**–prconf**] [**–defaults**]

/usr/sbin/ffbconfig [**–propt**] [**–prconf**]

/usr/sbin/ffbconfig [**–help**] [**–res** ?]

DESCRIPTION

ffbconfig configures the FFB Graphics Accelerator and some of the X11 window system defaults for FFB.

The first form of **ffbconfig** shown in the synopsis above stores the specified options in the OWconfig file. These options will be used to initialize the FFB device the next time the window system is run on that device. Updating options in the OWconfig file provides persistence of these options across window system sessions and system reboots.

The second and third forms which invoke only the **–prconf**, **–propt**, **–help**, and **–res** ? options do not update the OWconfig file. Additionally, for the third form all other options are ignored.

Options may be specified for only one FFB device at a time. Specifying options for multiple FFB devices requires multiple invocations of **ffbconfig**.

Only FFB-specific options can be specified through **ffbconfig**. The normal window system options for specifying default depth, default visual class and so forth are still specified as device modifiers on the openwin command line (see **Xsun**(1)).

The user can also specify the OWconfig file that is to be updated. By default, the machine-specific file in the /etc/openwin directory tree is updated. The **–file** option can be used to specify an alternate file to use. For example, the system-global OWconfig file in the /usr/openwin directory tree can be updated instead.

Both of these standard OWconfig files can only be written by root. Consequently, the **ffbconfig** program, which is owned by the root user, always runs with setuid root permission.

OPTIONS

–dev *device-filename*
Specifies the FFB special file. The default is **/dev/fbs/ffb0**.

–file machine | **system**
Specifies which OWconfig file to update. If machine, the machine-specific OWconfig file in the /etc/openwin directory tree is used. If **system**, the global OWconfig file in the /usr/openwin directory tree is used. If the file does not exist, it is created.

–res *video-mode* **[now | try [noconfirm | nocheck]]**

Specifies the video mode used to drive the monitor connected to the specified FFB device.

The format of these built-in video modes is:

widthx**height**x**rate**

where **width** is the screen width in pixels, **height** is the screen height in pixels, and **rate** is the vertical frequency of the screen refresh. The **s** suffix of 960x680x112s and 960x680x108s means that these are stereo video modes. The **i** suffix of 640x480x60i and 768x575x50i designates interlaced video timing. If absent, non-interlaced timing will be used. As a convenience, –**res** also accepts formats with @ in front of the refresh rate instead of x. For example: 1280x1024@76. The list of valid video-modes is show below. This list can also be obtained by running **ffbconfig** with the –**res** ? option (the third form shown in the command synopsis above).

NAME

pmconfig – Configure the power management system

SYNOPSIS

/usr/sbin/pmconfig

/usr/sbin/pmconfig [**–r**]

AVAILABILITY

SUNWpmu

DESCRIPTION

pmconfig enables the current system autoshutdown information to be viewed and / or the power management configuration modified. **pmconfig** reads in the configuration file **power.conf**(4) and issues commands to make this power configuration active. This may involve commands to the power management pseudo driver (**pm**(7)) or a signal to the power daemon (**powerd**(1M)). If no daemon is present and autoshutdown information is present, a daemon will be started.

ERRORS

If the program cannot open either the pseudo driver or the configuration file it prints an error message to standard error. If the program encounters a syntax error in the configuration file, it prints an error message and the line number of the error in the configuration file. It then skips the rest of the information on that line and processes the next line. Any configuration information already processed on the line containing the error is *used.*

All error messages start with "pmconfig (line n): ", and may be followed by:

Can't find device name :
The first field is not a device name.

Can't find threshold value :
the field following the device name is not an integer.

Too many threshold values :
More idle times than the device supports were given.

Unrecognizable dependent name :
The dependent field is not a device name.

a standard error message
Returned from the pm driver.

OPTIONS

–r reset all power managed devices to unconfigured

FILES

/etc/power.conf
system power management configuration file

SEE ALSO

pm(7), **power.conf**(4), **powerd**(1M)

NAME

powerd – power manager daemon

SYNOPSIS

/usr/lib/power/powerd [–n]

AVAILABILITY

SUNWpmu

DESCRIPTION

This daemon manages two types of system shutdown. The two types of shutdown are automatic shutdown, set on a daily basis, and low power shutdown on systems which supports battery operation. If the system suspend module, **cpr**(7), is present, it will be used to shut the system down, otherwise the **poweroff**(1M) utility will be used. The auto-shutdown information is read from the file **/etc/power.conf** by the daemon. It is reread whenever the daemon receives a hangup signal, SIGHUP.

Automatic shutdown can occur when two conditions are met. The current time is between the start and finish times, and the system has been idle for at least the set time period. System idleness is determined by inactivity on the console keyboard and mouse.

The start and finish times are specified in the file in 24-hour time notation, measured since the start of the day (12:00 am). If the finish time is less than or equal to the start time, the active period of the daemon will span from midnight to the finish time and from the start time to the following midnight. Thus to specify continuous operation, the finish time may be set equal to the start time. Specifying a negative idle time, disables automatic shutdowns from occurring.

Low power shutdown will occur if the system is running from battery and the daemon monitors that the charge in the battery is too low to reliably continue operation.

Immediately prior to system shutdown, the daemon notifies **syslogd**(1M) of the shutdown, which broadcasts the notification.

OPTIONS

–n No broadcast mode. The daemon will shutdown the system silently without notifying **syslogd**(1M).

FILES

/etc/power.conf
used to obtain the current daemon autoshutdown settings

NOTES

The daemon uses shared memory IPC, which may increase the system image size if the shared memory module has not already been loaded.

The daemon ensures that only one daemon is running. If another daemon is running, then the new daemon will exit with an error. If the daemon dies unexpectedly (non-maskable signal) then residual shared memory state will remain. Starting a new daemon will remove this residual state.

SEE ALSO

cpr(7), **pm**(7), **pmconfig**(1M), **power.conf**(4), **poweroff**(1M), **syslogd**(1M)

NAME

prtdiag – print system diagnostic information

SYNOPSIS

/usr/platform/sun4d/sbin/prtdiag [–**v**] [–**l**]

AVAILABILITY

SUNWkvm.d
SUNWkvm.u

DESCRIPTION

prtdiag displays system configuration and diagnostic information.

The diagnostic information lists any failed Field Replaceable Units (FRUs) in the system.

prtdiag is supported only on sun4d and sun4u machines.

The interface, output, and location in the directory hierarchy for **prtdiag** are uncommitted and subject to change in future releases.

OPTIONS

The following options are supported:

–**v** verbose mode.
On sun4d machines, displays the time of the most recent AC power failure, and the most recent system watchdog information. This information is useful only to depot repair and manufacturing for detailed diagnostics of FRUs.

On sun4u machines, displays various revision information, and (if applicable) environmental status, most recent AC power failure, and the most recent hardware fatal error information.

–**l** log output.
If failures or errors exist in the system, output this information to **syslogd(1M)** only.

EXAMPLES

The example below displays sample output from a SPARCcenter 2000 machine.

```
example% /usr/platform/sun4d/sbin/prtdiag

System Configuration:  Sun Microsystems  sun4d SPARCcenter 2000
System clock frequency: 40 MHz
Memory size: 448Mb
Number of XDBuses: 2
```

NAME

sm_configd – Solstice SyMON configuration reader

SYNOPSIS

/opt/SUNWsymon/sbin/sm_configd [**–D** *debug-value*] [**–T** *file*] [**–i** *interval*]

AVAILABILITY

SUNWsymon

DESCRIPTION

Monitors the physical configuration of a machine and reports on the status of components. For further details, please see the *Solstice SyMON User's Guide.*

OPTIONS

–D Set a debug option for AIL.

–T Run the configuration from a file; for testing purposes.

–i Set the polling interval for the Config Reader.

FILES

May use an optional file for test purposes.

SEE ALSO

symon(1), **sm_egd**(1M), **sm_logscand**(1M), **sm_krd**(1M), **sm_symond**(1M)

NAME

sm_egd – Solstice SyMON event generator

SYNOPSIS

/opt/SUNWsymon/sbin/sm_egd [**–i** *interval*] [**–d** *debug-level*]
[**–h** *log-file*] [**–H** *event-history-file*] [**–R** *rules-file*] [**–I** *init-file*]
[**–l** *shared-object* **–f** *shared-function*] [**–r** *export-root*]
[**–D** *AIL-debug-value*] [**–B** *event-directory*] [**–t** *target-machine*]
[**–S**] [**–L** *Tcl-directory*] [**–U** *username*]

AVAILABILITY

SUNWsymon

DESCRIPTION

Monitors other **symon** agents and reports events based on Tcl rules defined in rules files.

OPTIONS

–i Specify the polling interval for examining agents.

–d Specify a flag for debugging the event generator.

–h Specify a logfile location.

–H Specify a file to save event numbers across invocations.

–R Specify a rules file.

–I Specify an initialization file.

–l Specify a shared object to be loaded.

–f Specifies the function in a shared object to be run after loading.

–r Specifies the name of the root for the exported data.

–D Specifies an AIL debugging flag.

–B Specifies the directory for retaining events.

–t Specifies the machine to be monitored.

–S Allows core dumps on failure.

–L Specifies the location of the Tcl directory.

–U Specifies a user name under which to run the program.

FILES

rules.tcl Specifies the rules, in Tcl, for the event generator. Located in **/etc/opt/SUNWsymon.**

event_gen.tcl The initialization file for the event generator. Located in **/etc/opt/SUNWsymon.**

event_log The log file for events. Located in **/var/opt/SUNWsymon/***target***.**

EG_events Stores the last event number. Located in **/var/opt/SUNWsymon/***target***.**

events/*∗* Each event in the all events hierarchy. Located in **/var/opt/SUNWsymon/***target***.**

SEE ALSO

symon(1), **sm_krd**(1M), **sm_logscand**(1M), **sm_configd**(1M), **sm_symond**(1M)

NAME

sm_krd – Solstice SyMON kernel reader

SYNOPSIS

/opt/SUNWsymon/sbin/sm_krd [**–d**] [**–D** *AIL-debug-flag*] [**–v**]
[**–t**] [**–r**] [**–R**] [**–U** *kernel-file*] [**–M** *kmem-file*] [**–S** *swap-file*]
[**–i** *interval*] [**–P** *count*] [**–T**] [*count*]

AVAILABILITY

SUNWsymon

DESCRIPTION

sm_krd monitors the kernel on an active machine, and reports data to clients. For more information, please see the *Solstice SyMON User's Guide.*

OPTIONS

–d Activate Kernel Reader debugging.

–D Specify an AIL debugging level (values can be added together for combinations of debug output):
1=print AIP version
2=list of hierarchy updates
4=trace requests and connections
8=tell if replacing an existing node
10=debug pruning
20=trace memory use
40=report **sm_symond** traffic
80=sleep 30 seconds before starting
100=fake server death if **/tmp/dead** exists

–v Run the kernel reader in verbose mode.

–t Set the timer flag.

–r Set the resource information flag.

–R Set the resource information summary flag.

–U Specify the name of the kernel file.

–M Specify the name for the kmem file.

–S Specify the name of the swap file.

–i Specify the polling interval.

–P Run for the specified number of intervals, then quit.

–T Build the tree for debugging.

count Automatically report data for every *count* intervals.

SEE ALSO

symon(1), **sm_egd**(1M), **sm_logscand**(1M), **sm_configd**(1M), **sm_symond**(1M)

NAME

sm_logscand – Solstice SyMON log file scanner

SYNOPSIS

/opt/SUNWsymon/sbin/sm_logscand [**–i** *interval*] [**–L** *TCL-library*] [**–U** *user-name*] *log-definition-file*

AVAILABILITY

SUNWsymon

DESCRIPTION

Scans the log files, as described in the log definition file.

OPTIONS

–i Set the polling interval to update log files.

–L Specify the location of the Tcl library.

–U Specify a user name for running the program.

FILES

log-definition-file Initialization file for the log scanner. Located in **/etc/opt/SUNWsymon.**

SEE ALSO

symon(1), **sm_egd**(1M), **sm_krd**(1M), **sm_configd**(1M), **sm_symond**(1M)

NAME

sm_symond – Solstice SyMON process controller

SYNOPSIS

/opt/SUNWsymon/sbin/sm_symond [**–n**] [**–d** *debug-level*] [**–D** *AIL-debug-level*] [**–p** *output-level*] [**–P** *minutes*] [**–i** *intervals*]

AVAILABILITY

SUNWsymon

DESCRIPTION

sm_symond is a tool to manage symon processes. Its primary role is to start the symon agents, monitor those agents for crashes, and provide RPC information to clients that wish to access any of those agents.

The primary repository for agent data is the file **/etc/opt/SUNWsymon/sm_symond.conf** (see **sm_symond.conf**(4)).

When **sm_symond** is run, it first reads **/etc/opt/SUNWsymon/sm_symond.conf** to determine the local agents to be spawned. It then spawns those agents. If an entry indicates that an agent may exist on a remote system, symond will poll that system looking for another symond to get information on that agent.

Symond serves a hierarchy of information via RPC to any requesting client. Each agent should produce a hierarchy that is readable.

sm_symond is also responsible for looking at the **auth_checker.tcl** and **auth_list.tcl** scripts to determine if a Solstice SyMON user has access to the symon data.

OPTIONS

–n　Do NOT dissociate process and child agents from terminal. Leave stdin, stdout, stderr open for output.

–d　Debugging level for **sm_symond.** These values can be added together for combinations of debug output:
1=trace
2=callbacks
4=rpc
8=spawn info
16=debug access control
32=config file info

–D　Debugging level for AIL for hierarchy transport.

–p　Print hierarchy level:
1=nodes
5=nodes and prop
10=nodes, prop, and data

–P　Turn on profiling to dump after specified number of minutes.

–i　Sampling interval for checking if the agents are still alive.

FILES

/etc/opt/SUNWsymon/sm_symond.conf
list of agents for invocation.

SEE ALSO

sm_configd(1M), **sm_egd**(1M), **sm_krd**(1M), **sm_logscand**(1M), **sm_symond.conf**(4), **symon**(1)

NOTES

sm_symond can only be run by root.

NAME

speckeysd – Detects special keys on Type 5 or Compact 1 keyboard

SYNOPSIS

/usr/openwin/bin/speckeysd

AVAILABILITY

SUNWpmow

DESCRIPTION

speckeysd(1M) is a daemon that is started at OpenWindows start time to pick up the Sun Special Key strokes from Type 5 and Compact 1 keyboards. The Sun Special Keys are the following:

Power Key
Shift-Power Key
RaiseVolume Key
RaiseBrightness Key
LowerVolume Key
LowerBrightness Key
Mute Key
Degauss Key

The daemon waits on the Sun Special Key strokes, which are sent to it by the X Windows server as XEvents. On receiving the keystrokes, the daemon will then fork off a service to handle the key.

If the Sun Special Key has been specified as a repeatable key, then a pipe is opened to the service's STDIN. Every subsequent keystroke that is received within a timeout is sent to the service through the pipe as the character 'k'.

The daemon reads speckeysd.map(4), a keys-to-service map file, to determine which of the Sun Special Keys to expect and what service to spawn off to handle the key stroke.

FILES

/usr/openwin/lib/speckeysd.map	keys-to-service map file
/tmp/speckeysd.lock	lock-file generated by the daemon

SEE ALSO

speckesyd.map(4)

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

sys-suspend – Suspend the system and power off

SYNOPSIS

/usr/openwin/bin/sys-suspend [–**fnx**]

AVAILABILITY

SUNWpmow

DESCRIPTION

sys-suspend(1M) invokes the **uadmin**(1M) system call with the right options to suspend the whole system. A system can be suspended to conserve power or to prepare the system for transport. It should not be used in place of a shutdown when performing any hardware reconfiguration or replacement.

The current system state will be preserved until a resume operation is performed (the next power on).

On a resume from a manually initiated suspend in the windows environment, the system brings up **xlock**(1) to make certain that only the same person who suspended the system can have access to the system. In a non-windows environment, the user will be prompted for password. If the suspend was initiated by the **powerd**(1M), a. k. a. **AutoShutdown,** mechanism, no additional security measure is initated. It is the user's responsibility to secure his / her work session before **AutoShutdown** takes place.

It is possible that when devices or processes are performing critical or time sensitive operations (such as real time operations) the system may fail to suspend. When this occurs, the system will remain in its current running state. Messages reporting the failure will be displayed on the console. Once the system is successfully suspended the resume operation will always succeed barring external influences such as hardware reconfiguration or the like.

OPTIONS

–**f** Force suspend. This should be used with care. Using this option causes the system to force stops all processes that does not through the default mechnism. This option should be used only during unattended operations.

–**n** Disable confirmation. This flag disables the confirmation popup dialog at suspend time.

–**x** Disable lockscreen. This flag disables the execution of lockscreen at resume time.

FILES

/kernel/misc/cpr	loadable module for cpr
/cprboot	special bootstrapper for cpr
/.CPR	system state file
/.cpr_generic_info	sys-suspend control file
/.cpr_defaultboot_info	sys-suspend control file
/etc/default/sys-suspend	file for setting a default value for the PERM environment variable. PERM determines who are allowed to use this command. Allowed values are:

all	everybody can use this command (default)
-	nobody can use this command
<user1, user2, etc.>	a user in this user list can use this command

NOTES

xlock(1) on resume can be disabled by default. The following line needs to be added to the user's **.Xdefaults** or **.OWdefaults** file:

```
Syssuspend*xlock:        False
```

The xlock mode defaults to *life*. This can be changed by adding the following line to the user's **.Xdefaults** or **.OWdefaults** file:

```
Syssuspend*xlockmode:          <xlockmode>
```

SEE ALSO

uadmin(2), **cpr**(7)

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 shelltool and commandtool.

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 (for example, **set term=sun-cmd**

c. Set the values of columns and rows to the value noted above.

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

vtsui.ol – SunVTS Graphic User Interface (OpenLook)

SYNOPSIS

vtsui.ol [**–qv**] [**–h** *hostname*]

AVAILABILITY

SUNWvts

DESCRIPTION

The **vtsui.ol** command starts up the OpenLook version of **SunVTS** graphic user interface. There can be multiple instances of **vtsui.ol** 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 graphic user interface window.

vtsui.ol can be 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. **vtsui.ol** is also automatically started up by the **sunvts**(1M) command.

See the "SunVTS User's Guide" for a complete description on using the graphic 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**(1M), **vtstty**(1M), **vtsprobe**(1M)

NAME

power.conf – power management configuration information file

SYNOPSIS

/etc/power.conf

AVAILABILITY

SUNWpmr

DESCRIPTION

The **power.conf** file is used by the power management configuration program, **pmconfig**(1M), to initialize the settings for power management of the system.

There are two types of entries in the **power.conf** file, **device management** entries and **system management** entries. These two types of entries are described in the corresponding sections below.

DEVICE MANAGEMENT

Devices not appearing in this file will not be power managed without explicit configuration using the power management pseudo driver (see **pm**(7D)). It is recommended the power management framework be fully understood before modifying device management entries in this file. Although inappropriate settings will not cause system damage, severe performance reduction may result.

Device management entries consist of line by line listings of the devices to be configured. Each line is of the form:

device_name *threshold* . . . *dependents* . . .

Each line must contain a *device_name* field and a *threshold* field; it may also contain a *dependents* field. The fields must be in that order (*device_name*, *threshold*, *dependents*). Fields and sub-fields are separated by white space (tabs or spaces). A line may be more than 80 characters. If a newline character is preceded by a backslash ('\') it will be treated as white space. Comment lines must begin with a hash character ('#').

The *device_name* field specifies the device to be configured. *device_name* is either a pathname specifying the device special file or a "relative" pathname containing the name of the device special file. When using the latter format, instead of using the full pathname, it is possible to omit the portion of the pathname specifying the parent devices. This includes the leading '/'. Using this "relative" pathname format, the first device found with a full pathname containing *device_name* as its tail is matched. In either case, the leading **/devices** component of the pathname does not need to be specified.

For example, a SCSI disk target with the following full path name:

/iommu@f,e000/sbus@f,e001/espdma@f,4000/esp@f,8000/sd@1,0

may also be specified as:

sbus@f,e000/espdma@f,4000/esp@f,8000/sd@1,0

or

esp@f,8000/sd@1,0

or

sd@1,0

The *threshold* field is used to configure the power manageable components of a device. These components represent entities within a device which may be power managed separately. This field may contain as many integer values as the device has components. Each *threshold* time specifies the idle time in seconds before the respective component may be powered down. If there are fewer component *threshold* times than device components, the remaining components are not power managed. To explicitly disable power down for a component use a value of **−1**. At least one component *threshold* must be specified per device (in the file).

The *dependents* field may contain a list of *logical* dependents for this device. A *logical* dependent is a selected device that is not physically connected to the power managed device (e.g. the display and the keyboard). A dependent device is one which must be idle and powered down before the managed device may be powered down. The *dependents* field entries use the same formats allowed in the first field and are separated by white space. A device must previously have been configured before it may be used as a dependent.

SYSTEM MANAGEMENT

The system management entries control power management for the system as a whole. They are distinguished by the use of the special device names below.

Note that the following (**autoshutdown**) entry is not intended to be hand edited, but to be maintained by **dtpower**(1M).

If the *device_name* field contains the special device name "**autoshutdown**", the *threshold* value specifies the *system idle time* (measured as discussed below) before the system may be shut down by **powerd**(1M). The *threshold* value is followed by *start* and *finish* times (each in the format hh:mm) which specify the time period during which the system may be automatically shut down (see **powerd**(1M)). Following the *start* and *finish* times is the *behavior* field, consisting of one of the words **shutdown**, **noshutdown**, **autowakeup**, or **default.**

If the *behavior* field is **shutdown** then the system will be automatically shut down when it has been idle for the number of minutes specified in the *threshold* value and the time of day falls between the *start* and *finish* values.

If the *behavior* field is **noshutdown** then the system is never automatically shut down.

If the *behavior* field is **autowakeup** and the hardware has the capability to do autowakeup, then the system is shut down as if the value were **shutdown** and the system will be restarted automatically the next time that the time of day equals the *finish* time.

If the *behavior* field is **default** then the behavior of the system will depend upon which model it is. Desktop models which were first put into production after October 1, 1995 will behave as if the *behavior* field were set to **shutdown** and desktop models first put into production before this date and server models will act as if the *behavior* field were set to noshutdown. The determination of default behavior is made by looking for the existence of a root node property named energystar-v2.

If the *device_name* field contains the special device name "**ttychars**", the *threshold* field will be interpreted as the maximum number of tty characters which may pass through the ldterm module and the system still be considered to be idle. If no entry is provided this value defaults to 0.

If the *device_name* field contains the special device name “**loadaverage**”, the (floating point) *threshold* field will be interpreted as the maximum load average that may be seen and the system still be considered to be idle. If no entry is provided this value defaults to 0.04.

If the *device_name* field contains the special device name “**diskreads**”, the *threshold* field will be interpreted as the maximum number of disk reads which may be done by the system and the system will still be considered to be idle. If no entry is provided this value defaults to 0.

If the *device_name* field contains the special device name “**nfsreqs**”, the *threshold* field will be interpreted as the maximum number of NFS requests which may be sent or received by the system and it still be considered to be idle. Null requests, access requests and gettattr requests are excluded from this count. If no entry is provided this value defaults to 0.

The values for tty characters, disk reads and NFS requests are determined by periodic sampling of the kstat interface. The thresholds for these events apply to a period extending into the past for *system idle time* minutes as specified in the “**autoshutdown**” entry described above.

The value for load average is also determined by periodic sampling of the kstat interface. The threshold for this value is an instantaneous one. The system won’t be considered idle with respect to load average until *system idle time* minutes have passed with the sampled load average value not exceeding the threshold.

If the *device_name* field contains the special device name “**idlecheck**”, the *device_name* field must be followed by the pathname of a program to be executed to determine if the system is idle. If autoshutdown is enabled and the console keyboard, mouse, tty, CPU (as indicated by load average), network (as measured by NFS requests) and disk (as measured by read activity) have been idle for the amount of time specified in the *autoshutdown* entry specified above, and the time of day falls between the *start* and *finish* times, then this program will be executed to check for other idleness criteria. The value of the idle time specified in the above *autoshutdown* entry will be passed to the program in the environment variable PM_IDLETIME. The process must terminate with an exit code which represents the number of minutes that the process considers the system to have been idle.

There is no default idlecheck entry. The default behavior is to consider only mouse, keyboard, tty, load average, NFS requests and disk reads as indicators of non-idleness. To extend the definition of non-idleness a shell script can be created which must exit with the number of minutes it considers the system to have been idle by its criteria. The path to this new script can then be put in the idlecheck entry in **power.conf.**

EXAMPLES

The following is a sample **power.conf** file.

```
# This is a sample power management configuration file
# Fields must be separated by white space.
#
```

```
# Name            Threshold(s)   Logical Dependent(s)
/dev/kbd          1800
/dev/mouse        1800
/dev/fb           0 0              /dev/kbd /dev/mouse

#Example of a second display
/dev/fb1          0 0              /dev/kbd /dev/mouse

# This entry is maintained by dtpower(1M)
# This (default as of SunOS 2.5) entry causes the system  to be shut down
# after 30 minutes of idle time if it is a model first shipped after
# Oct 1, 1995.  Older models default to noshutdown.
#
#                                        autoshutdown in effect
# Auto-Shutdown       Idle(min)       Start/Finish(hh:mm)     Behavior
autoshutdown          30        9:00 9:00                 default

# Idlecheck  program is passed autoshutdown idle time entry in $PM_IDLETIME
# returns number of minutes the system has been idle in exit code
idlecheck    /home/critical/idlecheck
```

The following is a sample idlecheck script.

```
#!/bin/sh
# This is a sample idlecheck script which considers the system not idle
# if user critical is logged in

critical=‘who | grep -w critical‘
if [ "$critical" ]                   # if "$critical" is not null string
then
        exit 0                       # not idle because critical logged in
else
        exit $PM_IDLETIME            # idle long enough
fi
```

SEE ALSO

dtpower(1M), **pmconfig**(1M), **powerd**(1M), **pm**(7D)

NAME

sm_symond.conf – agent listing for sm_symond

DESCRIPTION

The file **/etc/opt/SUNWsymon/sm_symond.conf** controls process spawning by **sm_symond.** The processes most typically dispatched by **sm_symond** are symon agents.

The **sm_symond.conf** file is composed of entries that either list an agent and its arguments, or specify agents to run on remote machines.

Local agents are listed, one per line, with the normal command line arguments, and are invoked by sm_symond. Remote agent entries have the following format:

> *host*:*agent-type*

Each entry is delimited by a newline. Comments may be inserted in the **sm_symond.conf** file by starting the line with a #.

The remote agent fields are:

host The name of the remote host where the agent is to be run.

agent-type The specific type of symon agent being run. Currently, the only agent type supported on remote machines is **EventGenerator.**

SEE ALSO

sm_symond(1M), **symon**(1), **sm_egd**(1M), **sm_krd**(1M), **sm_configd**(1M), **sm_logscand**(1M)

NAME

speckeysd.map – Sun Special Keys to service map file for speckeysd

SYNOPSIS

/usr/openwin/lib/speckeysd.map

AVAILABILITY

SUNWpmow

DESCRIPTION

The **speckeysd.map** file is used by the **speckeysd**(1M) daemon to determine which Sun Special Keys to look for in the X Windows environment, and which service to spawn off to handle the keys.

The file is composed of entries for Sun Special Keys that are position-dependent and have the following format:

```
Sun Special Key Keysym        Repeatable      Service
```

Each entry is delimited by a newline. Each field is delimited by white-space (either a space or a tab). The whole entry must come before a newline, ie. you cannot extend lines by putting a backslash () preceding the newline.

The fields are:

Sun Special Key Keysym — Which Sun Special Key keysym should the **speckeysd**(1M) look for? Each key has a keysym associated with it in the X Windows environment. The Sun Special Keys and the Keysyms associated with them are:

Degauss Key	SunVideoDegauss
Mute Key	SunAudioMute
LowerVolume Key	SunAudioLowerVolume
LowerBrightness Key	SunVideoLowerBrightness
RaiseVolume Key	SunAudioRaiseVolume
RaiseBrightness Key	SunVideoRaiseBrightness
Power Key	SunPowerSwitch
Shift-Power Key	SunPowerSwitchShift

Repeatable — Is the Sun Special Key that **speckeysd (1m)** is supposed to look for repeatable? The valid options are:

r	the key is repeatable
-	the key is not repeatable

Service — Which service should be spawned off if one of the Sun Special Keys are pressed and what arguments should be passed to it? The service field is always considered to be everything after the Repeatable field and white-spaces following it to the newline character. To ensure that there are no PATH issues, specify the service with the complete path.

Comments are allowed in the file. However, the comments are full line entries, from an initial hash character (#) to the newline.

EXAMPLES

The following is a sample speckeysd.map file.

```
# This is the special keys service map file.
#
# This file will let speckeysd know what special keys (represented by X
# Windows Keysyms) to expect and what services to spawn off to handle the
# keys.
#
SunVideoRaiseBrightness         r    $OPENWINHOME/bin/contrast -k -u 1
SunVideoLowerBrightness         r    $OPENWINHOME/bin/contrast -k -d 1
SunPowerSwitch                  -    $OPENWINHOME/bin/sys-suspend
SunPowerSwitchShift        -    $OPENWINHOME/bin/sys-suspend -n
```

NOTES

If the file is changed and the system is already in X Windows, the **speckeysd (1M)** daemon must be restarted to pick up the changes.

SEE ALSO

speckeysd(1M)

NAME

cpr – Suspend and resume module

SYNOPSIS

/kernel/misc/cpr

AVAILABILITY

SUNWcpr

DESCRIPTION

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

The principle way to suspend the system using this module is through the **syssuspend**(1M) command. There are other utilities which may be installed on your system which will also access this module (such as **uadmin**(1M), **uadmin**(2), or the *Power* key and the *Shift+Power* key on a type 5 keyboard).

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

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

In some cases the **cpr** module may be unable to perform the suspend operation. If a system contains additional devices outside the standard shipped configuration, it is possible that these additional devices may not support **cpr.** In this case, the suspend will fail and an error message will be displayed to that effect. These devices must be removed or its device drivers unloaded for suspend to work. Contact the device manufacturer to obtain a new version of device driver that supports **cpr.** A suspend may also fail when devices or processes are performing critical or time sensitive operations (e. g. real time operations). In this case the system will remain in its current running state. Messages reporting the failure will be displayed on the console and status returned to the caller. Once the system is successfully suspended the resume operation will always succeed barring external influences such as hardware reconfiguration or the like.

Some network based applications may fail across a suspend and resume cycle. This largely depends on the underlying network protocol and the applications involved. In general, applications that retry and automatically reestablish connections will continue to operate transparently on resume, those applications that do not, will likely fail.

The speed of suspend and resume can range from 15 seconds to a few minutes depending on the system speed, memory size and load. The typical time is around a minute.

FILES

/cprboot	special bootstrapper for cpr
/.CPR	system state file
/.cpr_generic_info	sys-suspend control file
/.cpr_defaultboot_info	sys-suspend control file

BUGS

The signals SIGFREEZE and SIGTHAW are not properly implemented for the Solaris 2.4 release, it will be available in a later release. This should only be a concern for specially customized applications that need to perform additional tasks at suspend or resume time, which none exists at the present time.

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

<ok> *set-default boot-file*

This resets the system and on the next power on the system will boot normally.

NOTES

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

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

SEE ALSO

sys-suspend(1M), **uadmin**(1M), **uadmin**(2)

NAME

ecpp – IEEE 1284 ecp, nibble and centronics compatible parallel port driver

SYNOPSIS

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

fd = open("/dev/ecpp0", flags);

DESCRIPTION

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

Default Operation

Each time the ecpp device is opened, the device is marked as EBUSY and the configuration variables are set to their default values. The write_timeout period is set to 60 seconds. The driver sets the mode variable according to the following algorithm: The driver initially attempts to negotiate the device into ECP mode. If this should fail, the driver will attempt to negotiate into Nibble mode. If Nibble mode negotiation should fail, the driver will operate in Centronics mode. The application may attempt to negotiate the device into a specific mode or set the write_timeout values through the **ECPPIOC_SETPARMS ioctl**(2) call. In order for the negotiation to be successful, both the host workstation and the peripheral must support the requested mode.

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

Read/Write Operation

ecpp is a full duplex STREAMS device driver. While an application is writing to an IEEE 1284 compliant device, another thread may read from it. **write**(2) will return when all the data has been successfully transferred to the device.

Write Operation

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

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

Read Operation

If a failure or error condition occurs during a **read**(2), the number of bytes successfully read is returned (short read). When attempting to read the port that has no data currently available, **read**(2) returns 0 if O_NDELAY is set. If O_NONBLOCK is set, **read**(2) returns -1 and sets errno to EAGAIN. If O_NDELAY and O_NONBLOCK are clear, **read**(2) blocks until data become available.

IOCTLS

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

ECPPIOC_GETPARMS

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

ECPPIOC_SETPARMS

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

Transfer Parameters Structure

This structure is defined in **<sys/ecppio.h>.**

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

The **write_timeout** field is set to **ECPP_W_TIMEOUT_DEFAULT.** The **write_timeout** field specifies how long the driver will wait for the peripheral to respond to a transfer request. The value must be greater than 0 and less than ECPP_MAX_TIMEOUT. Any other values are out of range.

The **mode** field reflects the IEEE 1284 mode that the parallel port is currently configured to. The mode may be set to only one of the following bit values.

```
#define ECPP_CENTRONICS        0x1
#define ECPP_COMPAT_MODE       0x2
#define ECPP_NIBBLE_MODE       0x3
```

```
#define ECPP_ECP_MODE          0x4
#define ECPP_FAILURE_MODE      0x5
```

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

BPPIOC_TESTIO

Tests the transfer readiness of ECPP_CENTRONICS or ECPP_COMPAT_MODE devices.
If the current mode of the port is ECPP_CENTRONICS or ECPP_COMPAT_MODE, this command determines if write (2) would succeed. If it is not one of these modes, EINVAL is returned. BPPIOC_TESTIO determines if a **write**(2) would succeed by checking the open flag and status pins. If any of the status pins are set, a transfer would fail. If a transfer would succeed, zero is returned. If a transfer would fail, -1 is returned, and **errno** is set to **EIO** , and the state of the status pins is captured. The captured status can be retrieved using the **BPPIOC_GETERR ioctl (2)** call. Note that the **timeout_occurred** and **bus_error** fields will never be set by this **ioctl**(2)**. BPPIOC_TESTIO** and **BPPIOC_GETERR** are compatible to the ioctls specified in **bpp**(7). However, **bus_error** is not used in this interface.

BPPIOC_GETERR

Get last error status.
The argument is a pointer to a **struct bpp_error_status**. This structure is described below. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a **write**(2) call, or the status of the bits at the most recent **BPPIOC_TESTIO ioctl**(2) call.

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

pin_status indicates possible error conditions under ECPP_CENTRONICS or ECPP_COMPAT_MODE. Under these modes, the state of the status pins will indicate the state of the device. For instance, many Centronics printers lower the nErr signal when a paper jam occurs. The behavior of the status pins depends on the device. As defined in the IEEE 1284

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

Error Status Structure

struct bpp_error_status is defined in the include file **<sys/bpp_io.h>.** The valid bits for **pin_status** are presented below. A set bit indicates that the associated pin is asserted. For example, if BPP_ERR_ERR is set, nErr is asserted.

```
struct    bpp_error_status {
   char   timeout_occurred; /* 1=timeout */
   char   bus_error;        /* not used */
   u_char pin_status;    /*
                                 * status of pins
                         * which could cause
                         * error.
                         */
};

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

ERRORS

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

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

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

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

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

ENXIO The driver has received an open request for a unit for which the attach failed. The driver has received a write request for a unit which has an active peripheral error.

FILES

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

SEE ALSO

ioctl(2), **read**(2), **write**(2), **streamio**(7)

NAME

mic – Multi-interface Chip driver

SYNOPSIS

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

open("/dev/term/mic/a", *mode***);**
open("/dev/term/mic/b", *mode***);**
open("/dev/term/mic/ir", *mode***);**

AVAILABILITY

SUNWmic

PLATFORM

SPARCstation Voyager

DESCRIPTION

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

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

The **mic** module is a loadable STREAMS driver that provides basic support for the MIC hardware, together with basic asynchronous communication support. The driver supports those **termio**(7) device control functions specified by flags in the **c_cflag** word of the **termios** structure, excluding HUPCL, CLOCAL, CIBAUD, CRTSCTS and PAREXT. The driver does not support device control functions specified by flags in the **c_iflag** word of the **termios** structure. Specifically, the driver assumes that IGNBRK and IGNPAR are always set. All other **termio**(7) functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the **ldterm**(7) and **ttcompat**(7) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(7) interface.

The infra-red port provides access to two different modes of modulation. The default mode is called pulse mode and is compatible with the Infra-red Data Association (IrDA) modulation and the Hewlett-Packard Serial Infra-red (SIR) modulation. The second modulation is called high frequency mode and is compatible with the Sharp Amplitude Shift Keying (ASK) modulation. The default modulation when using high frequency mode is 500 KHz.

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

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

IOCTLS

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

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

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

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

MIOCGETM_IR
Returns the current IR mode defined in micio.h

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

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

carrier frequency = 19660 / (4 (divisor +1)) KHz

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

divisor = 19660 / frequency / 4 – 1

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

MIOCCLPBK_IR
Clears IR loopback mode.

There are two **mic** specific **ioctl( )** calls:

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

MIOCCLPBK
Clear SCC loopback mode.

ERRORS

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

ENXIO The unit being opened does not exist.

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

FILES

/dev/term/mic/a	asynchronous serial line using port a
/dev/term/mic/b	asynchronous serial line using port b
/dev/term/mic/ir	asynchronous serial infra-red line using the infra-red port

DIAGNOSTICS

mic: Rx FIFO overflow

The mic's internal 64 character buffer overflowed before it could be serviced.

mic: Rx buffer full - draining

The driver's character input buffer overflowed before it could be serviced.

NOTES

Currently hardware flow control is not implemented. The state of DCD, CTS, RTS and DTR interface signals cannot be queried, nor can hardware flow control be enabled using the CRTSCTS flag in the **c_cflag** word of the **termios** structure.

SEE ALSO

tip(1), **ports**(1M), **ioctl**(2), **open**(2), **ldterm**(7), **termio**(7), **ttcompat**(7),

NAME

pm – Power Management Driver

SYNOPSIS

#include <sys/pm.h>
int ioctl(int fildes, int command, int arg);

AVAILABILITY

SUNWpmu

DESCRIPTION

The Power Management driver provides an interface for applications to configure the devices within the system for power management. The interface is provided through **ioctl**(2) commands. The *pm* driver may be accessed using **/dev/pm.**

fildes is an open file descriptor that refers to the **pm** driver. *command* determines the control function to be performed as described below. *arg* 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

Unless configured by using the commands below, **pm** does not power manage devices by default. Note, however, that the **pmconfig**(1M) program is typically run at boot time, and by reading the **power.conf**(4) file will use the commands below to configure **pm.** Any devices configured for power management by **pm** will have their drivers loaded (if not already) and locked into memory until that device is unmanaged. Some devices may be able to fully operate at non–full power levels. Using the command **PM_SET_POWER** on such a device allows this low power mode to become the normal (on) power level for that device. This mode of operation is distinct from the power managed mode of operation.

pm periodically searches the system for devices which it can power manage. A device will only be power managed when it is not in use (explained further below). When a power managed device is subsequently used, it will be automatically returned to normal power.

The **pm** model of power management is to view the system as a collection of devices. Each device is a collection of components, a component is the smallest power manageable unit. The devices, and the components within those devices, which are power manageable are dependent upon the implementation of their respective device drivers. A power manageable component has three states. It may be *busy* (in use), it may be *idle* (not in use but using normal power), or it may be *power managed* (not in use and not using normal power). The **pm** driver manages the component transition from the second to the third state. **pm** uses two factors to determine this transition: the component must have been idle for at least the threshold time; and the device to which the component belongs must satisfy any dependencies requirements. A dependency is when a device requires another device to be power managed before it can be power managed. A device is considered to be power managed when all of its components are power managed. Note that dependencies occur on a per device basis: when a dependency exists, no components of a device may be managed unless all the components it depends upon are first managed. For more information, see the **Guide to Writing Device Drivers manual, attach**(9E), **detach**(9E), **power**(9E).

Thus the configuration of a device for power management is the setting of the threshold for any component that is to be managed and defining any dependencies for that device.

For all commands excluding **PM_SCHEDULE, arg** points to a structure of type *pm_request* defined in **sys/pm.h:**

```
typedef struct {
        char    *who;           /* device to configure */
        int     select;         /* selects the component or
                                   dependent of the device */
        int     level;          /* power or threshold level */
        char    *dependent;     /* hold name of dependent */
        int     size;           /* size of dependent buffer */
} pm_request;
```

The fields should contain the following data. *who* is a pointer to the name of the device to be configured. The name must be in the format described in **power.conf**(4). *select* is a non–negative integer specifying the component or dependent being configured. The numbering starts at zero. *level* is non–negative integer giving the threshold level in seconds or the desired power level. *dependent* is a pointer to a buffer which contains or receives the name of a device on which this device has a dependency. It uses the same format as the first field. *size* is the size of the dependent buffer.

Not all fields are used in each command. Upon error the commands will return **-**1, and set *errno* to the error condition specified below. The following error codes are common to all commands.

EFAULT: Bad address passed in as argument.

ENODEV:
Device is not power manageable, or device is not configured (Use **PM_SET_THRESHOLD** command first).

ENXIO: Invalid instance number (device not attached).

EPERM: Permission denied. You must be root or console owner.

PM_SCHEDULE:
arg sets the period in seconds of **pm** device scans. A value of zero inhibits scans which stops any further components from being managed. A negative value is ignored. The ioctl returns the new (or current) period.

PM_GET_IDLE_TIME:
Using the fields *who* and *select*, this command returns the time in seconds since the component was last busy. Error codes:

EINVAL: Device component out of range.

PM_GET_NUM_CMPTS:
Using the field *who,* this command returns the number of components defined for this device.

PM_GET_THRESHOLD:
Using the fields *who* and *select, this command returns the* threshold level of the component. Error codes:

EINVAL: Device component out of range.

PM_SET_THRESHOLD:
Using the fields *who, select* and *level,* this command sets the threshold level of the component. It returns zero on success. Error codes:

EINVAL: Device component out of range, or threshold value < 0.

PM_GET_POWER:
Using the fields *who* and *select,* this command returns the current normal power level of the component.

EINVAL: Device component out of range.

EIO: Non–power manageable device (or properties are removed).

PM_SET_POWER:
Using the fields *who, select* and *level,* this command sets the current normal power level of the component to the given power level.

EINVAL: Device component out of range, or power level <= 0.

EIO: Failed to power device or its parent or its dependents.

PM_GET_CUR_PWR:
Using the fields *who* and *select,* this command returns the current power level of the component.

EINVAL: Device component out of range.

PM_GET_NUM_DEPS:
Using the field *who,* this command returns the number of dependents configured for this device.

PM_GET_DEP:
Using the fields *who, select, level* and *dependent,* this command writes the name of dependent into the buffer supplied by the *dependent* field.

EINVAL: Dependent component out of range, or user buffer is too small for dependent name

EFAULT: Bad buffer address was given.

PM_ADD_DEP:
Using the fields *who* and *dependent,* this command adds the dependent to the device.

ENODEV: Dependent is non–power manageable or is not configured.

PM_REM_DEP:
Using the fields *who* and *dependent,* this command removes the dependent from the device.

ENODEV: Dependent is non–power manageable or is not configured, or the device has no dependents

PM_REM_DEVICE:
Using the field *who,* this command unmanages the device and returns the device to normal power, if it is not already.

PM_REM_DEVICES:
This command unmanages all devices and returns them to normal power.

NOTES

To unload a power managed driver, the driver must first be unmanaged using **PM_REM_DEVICE(S).**

Currently it is NOT an error to remove a nonexistent dependent or add a repeated dependent. The pseudo driver will silently ignore the redundant command.

SEE ALSO

intro(2), **ioctl**(2), **pmconfig**(1M), **power.conf**(4), **attach**(9E), **detach**(9E), **power**(9E)

NAME

pmc – Platform Management Chip driver

SYNOPSIS

#include <sys/pmcio.h>
int ioctl(int fildes, int command, int arg);

AVAILABILITY

SUNWpmc

PLATFORM

SPARCstation Voyager

DESCRIPTION

The Platform Management Chip driver provides a number of miscellaneous platform specific functions. Principally these are to provide power control for devices which cannot manage their own power control (see **ddi_power (9F)**) and to provide information about the connection status of the machine. Not all functions are supported on all platforms.

The user interface is provided through **ioctl (2)** commands. The **pmc** driver may be accessed using **/dev/pmc.** The system interface (to power manage devices) is provided by registering its power function (using the "platform-pm" property of the root node).

fildes is an open file descriptor that refers to the **pmc** driver. *command* determines the control function to be performed as described below. *arg* is not used and may be any value.

COMMAND FUNCTIONS

These functions fall into three categories: connection status, power control and miscellaneous. Connection status can be used to find out whether the following devices are plugged in: keyboard, ethernet and ISDN.

The power control function controls the removal of the platform power. Miscellaneous functions enable the reading of the digital to analog converter.

PMC_GET_KBD:
This command returns the connection status of the keyboard. When the keyboard is connected it will return **PMC_KB_STAT,** and zero when it is not connected.

PMC_GET_ENET:
This command returns the connection status of the ethernet. When the ethernet is connected it will return **PMC_ENET_STAT,** and zero when it is not connected.

PMC_GET_ISDN:
This command returns the connection status of the isdn channels. The return value is a bit map of the connected channels: **PMC_ISDN_ST0** for **NT,** **PMC_ISDN_ST1** for **TE.**

PMC_GET_A2D:
This command returns the result of an eight bit analog to digital conversion. The meaning of the reading is platform specific.

PMC_POWER_OFF:
This command is only available to the super-user. It turns off all power to the

system. Note that critical data may be lost if proper preparation prior to power removal is not performed.

The **poll(2)** interface is supported. It may be used to poll for connection status changes. A process wishing to detect such connection changes should use the **POLLIN** event flag. When ANY connection status changes, the **poll (2)** mechanism will be notified. It is up to the user to verify whether the connection status change is of interest.

ERRORS

EPERM Must be privileged user to use **PMC_POWER_OFF.**

SEE ALSO

ddi_power(9F), **intro**(2), **ioctl**(2), **open**(2), **pm**(7), **poll**(2)

NAME

fas – FAS SCSI Host Bus Adapter Driver

SYNOPSIS

fas@*sbus-slot*,0x8800000

AVAILABILITY

Limited to Sparc SBus-based systems with FAS366 based SCSI port, platforms and SBus SCSI Host Adapter options TBD.

DESCRIPTION

The **fas** Host Bus Adapter driver is a SCSA compliant nexus driver that supports the Qlogic FAS366 SCSI chip.

The **fas** driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing, wide and fast SCSI, almost unlimited transfer size (using a moving DVMA window approach), auto request sense but does not support linked commands.

Driver Configuration

The **fas** driver can be configured by defining properties in **fas.conf** which override the global SCSI settings. Supported properties are **scsi-options**, **target<***n***>-scsi-options**, **target<***n***>-sync-speed**, **target<***n***>-wide**, **target<***n***>-TQ**, **scsi-reset-delay**, **scsi-watchdog-tick**, **scsi-tag-age-limit**, **scsi-initiator-id**.

target<*n***>-scsi-options** overrides the **scsi-options** property value for **target<***n***>**. <*n*> can vary from 0 to f. The supported scsi-options are SCSI_OPTIONS_DR, SCSI_OPTIONS_SYNC, SCSI_OPTIONS_TAG, SCSI_OPTIONS_FAST, SCSI_OPTIONS_WIDE.

scsi-watchdog-tick is the periodic interval where the **fas** driver goes through all current and disconnected commands searching for timeouts.

scsi-tag-age-limit is the number of times that the **fas** driver attempts to allocate a particular tag ID that is currently in use after going through all tag IDs in a circular fashion. After finding the same tag ID in use **scsi-tag-age-limit** times, no more commands will be submitted to this target until all outstanding commands complete or timeout.

Refer to **scsi_hba_attach**(9F) for details.

EXAMPLES

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

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

This will disable tagged queuing, fast SCSI, and Wide mode for all **fas** instances. To disable an option for one specific **fas** (refer to **driver.conf**(4)):

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

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

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

The physical pathname of the parent can be determined using /devices tree or following the link of the logical device name:

```
# ls -l /dev/rdsk/c1t3d0s0
lrwxrwxrwx 1 root  other  78 Aug 28 16:05 /dev/rdsk/c1t3d0s0 ->
../../devices/iommu@f,e0000000/sbus@f,e0001000/SUNW,fas@3,8800000/sd@3,0:a,raw
```

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

```
SUNW,fas, instance #0
  ....
  Register Specifications:
    Bus Type=0x3, Address=0x8800000, Size=10
    Bus Type=0x3, Address=0x8810000, Size=40
```

Driver Capabilities

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

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

The target driver needs to enable **tagged-qing** and **wide-xfer** explicitly. The **untagged-qing** capability is always enabled and its value cannot be modified, because **fas** can queue commands even when **tagged-qing** is disabled.

Whenever there is a conflict between the value of scsi-options and a capability, the value set in scsi-options prevails. Only whom != 0 is supported in the **scsi_ifsetcap**(9F) call.

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

FILES

/kernel/drv/fas	ELF Kernel Module
/kernel/drv/fas.conf	Optional configuration file

SEE ALSO

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

NAME

ffb – 24-bit UPA color frame buffer and graphics accelerator

DESCRIPTION

ffb is a 24-bit UPA-based color frame buffer and graphics accelerator which comes in two configurations.

The single buffered frame buffer consists of 32 video memory planes of 1280 × 1024 pixels, including 24-bit single-buffering and 8-bit X planes.

The double buffered frame buffer consists 96 video memory planes of 1280 × 1024 pixels, including 24-bit double-buffering, 8-bit X planes, 28-bit Z-buffer planes and 4-bit Y planes. The driver supports the following frame buffer ioctls which are defined in **fbio**(7I).

> **FBIOPUTCMAP, FBIOGETCMAP, FBIOSVIDEO, FBIOGVIDEO,**
> FBIOVERTICAL, FBIOSCURSOR, FBIOGCURSOR, FBIOSCURPOS,
> FBIOGCURPOS, FBIOGCURMAX, FBIO_WID_PUT, FBIO_WID_GET

However, **ffb** does not support **FBIOGTYPE** which is part of **fbio**(7I).**The**replacement**is VIS_GETIDENTIFIER.**

FILES

/dev/fbs/ffb0	device special file

SEE ALSO

ffbconfig(1M), **mmap**(2), **fbio**(7I)

NAME

hme – SUNW,hme Fast-Ethernet device driver

SYNOPSIS

/dev/hme

DESCRIPTION

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

SUNW,hme

The **SUNW,hme** device provides 100Base-TX networking interfaces using SUN's **FEPS ASIC** and an Internal Transceiver. The FEPS ASIC provides the Sbus interface and MAC functions and the Physical layer functions are provided by the Internal Transceiver which connects to a **RJ-45** connector. In addition to the RJ-45 connector, an **MII** (Media Independent Interface) connector is also provided on all SUNW,hme devices except the SunSwith SBus adapter board. The MII interface is used to connect to an External Transceiver which may use any physical media (copper or fiber) specified in the 100Base-TX standard. When an External Transceiver is connected to the MII, the driver selects the External Transceiver and disables the Internal Transceiver.

The 100Base-TX standard specifies an "auto-negotiation" protocol to automatically select the mode and speed of operation. The Internal transceiver is capable of doing "auto-negotiation" with the remote-end of the link (Link Partner) and receives the capabilities of the remote end. It selects the **Highest Common Denominator** mode of operation based on the priorities. It also supports **forced-mode** of operation where the driver can select the mode of operation.

APPLICATION PROGRAMMING INTERFACE

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

hme and DLPI

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

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

- The maximum SDU is **1500** (**ETHERMTU -** defined in <sys / ethernet.h>).
- The minimum SDU is **0**.

- The **dlsap** address length is **8.**
- The MAC type is **DL_ETHER**.
- The **sap** length values is −**2** meaning the physical address component is followed immediately by a 2 byte **sap** component within the DLSAP address.
- The service mode is **DL_CLDLS**.
- No optional quality of service (QOS) support is included at present so the QOS fields are **0**.
- The provider style is **DL_STYLE2**.
- The version is **DL_VERSION_2**.
- The broadcast address value is Ethernet / IEEE broadcast address (**0xFFFFFF**).

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

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

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

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

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

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

hme Primitives

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

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

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

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

hme DRIVER

By default, the hme driver performs "auto-negotiation" to select the **mode** and **speed** of the link, when the Internal Transceiver is used.

When an External Transceiver is connected to the **MII** interface, the driver selects the External Transceiver for networking operations. If the External Transceiver supports "auto-negotiation", the driver uses the auto-negotiation procedure to select the link speed and mode. If the External Transceiver does not support auto-negotiation, it will select the highest priority mode supported by the transceiver.

The link can be in one of the *4* following modes:

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

These speeds and modes are described in the 100Base-TX standard.

The *auto–negotiation* protocol automatically selects:

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

The auto–negotiation protocol does the following:

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

The *internal transceiver* is capable of all of the operating speeds and modes listed above. When the internal transceiver is used, by *default*, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the Link Partner.

When an *external transceiver* is connected to the **MII** interface, the driver selects the external transceiver for networking operations. If the external transceiver supports auto-negotiation:

- The driver uses the auto-negotiation procedure to select the link speed and mode.

If the external transceiver *does not* support auto-negotiation

- The driver selects the highest priority mode supported by the transceiver.

Sometimes, the user may want to select the speed and mode of the link. The **SUNW,hme** device supports programmable **"IPG"** (Inter-Packet Gap) parameters **ipg1** and **ipg2**. By default, the driver sets **ipg1** to 8 **byte-times** and **ipg2** to 4 **byte-times** (which are the standard values). Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mpbs and accordingly, **IPG** will be set to 9.6 or 0.96 microseconds.

hme Parameter List

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

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

FILES

/dev/hme	**hme** special character device.
/kernel/drv/hme	System wide default device driver properties

SEE ALSO

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

Index