man pages section 9F: Kernel Functions for Drivers

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

Part No: 816–1062–10
November 2001

™
Adobe PostScript

011029@2471

Contents

Preface

Overview

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

Trusted Solaris Reference Manual

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

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

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

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

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

Man Page Sections

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

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

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

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

SYNOPSIS
: This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and

arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ]	The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
. . .	Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ' "`filename ...`" .
|	Separator. Only one of the arguments separated by this character can be specified at a time.
{ }	Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL
This section occurs only in subsection 3R to indicate the protocol description file.

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

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

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

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

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

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged

paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS

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

USAGE

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

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

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

ENVIRONMENT VARIABLES

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

EXIT STATUS

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

FILES

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

ATTRIBUTES

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

SUMMARY OF TRUSTED SOLARIS CHANGES
: This section describes changes to a Solaris item by Trusted Solaris software. It is present in man pages that have been modified from Solaris software.

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

DIAGNOSTICS
: This section lists diagnostic messages with a brief explanation of the condition causing the error.

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

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

BUGS
: This section describes known bugs and, wherever possible, suggests workarounds.

Introduction

NAME

Intro – introduction to DDI/DKI functions

DESCRIPTION

Section 9F describes the Solaris and Trusted Solaris kernel functions available for use by device drivers. See `Intro`(9) for how Trusted Solaris man pages are included in section 9F.

Note – The printed *Trusted Solaris 8 4/01 Reference Manual* includes only those man pages that have been modified or originate in the Trusted Solaris environment. Printed versions of unchanged `SunOS` 5.8 man pages are found in the *SunOS 5.8 Reference Manual*. For a fuller description, see Trusted Solaris Manual Page Display in `Intro`(1). The `SEE ALSO` man page heading has been subdivided to help users of the printed manual locate a referenced man page.

HEADINGS

In this section, the information for each driver function is organized under the following headings:

- `NAME` summarizes the function's purpose.
- `SYNOPSIS` shows the syntax of the function's entry point in the source code. `#include` directives are shown for required headers.
- `INTERFACE LEVEL` describes any architecture dependencies.
- `ARGUMENTS` describes any arguments required to invoke the function.
- `DESCRIPTION` describes general information about the function.
- `RETURN VALUES` describes the return values and messages that can result from invoking the function.
- `CONTEXT` indicates from which driver context (user, kernel, interrupt, or high-level interrupt) the function can be called.
- A driver function has *user context* if it was directly invoked because of a user thread. The `read`(9E) entry point of the driver, invoked by a `read`(2) system call, has user context.
- A driver function has *kernel context* if was invoked by some other part of the kernel. In a block device driver, the `strategy`(9E) entry point may be called by the page daemon to write pages to the device. The page daemon has no relation to the current user thread, so in this case `strategy`(9E) has kernel context.
- *Interrupt context* is kernel context, but also has an interrupt level associated with it. Driver interrupt routines have interrupt context.
- *High-level interrupt context* is a more restricted form of interrupt context. If `ddi_intr_hilevel`(9F) indicates that an interrupt is high-level, driver interrupt routines added for that interrupt with `ddi_add_intr`(9F) run in high-level interrupt context. These interrupt routines are only allowed to call `ddi_trigger_softintr`(9F) `mutex_enter`(9F) and `mutex_exit`(9F). Furthermore, `mutex_enter`(9F) and `mutex_exit`(9F) may only be called on mutexes initialized with the `ddi_iblock_cookie` returned by `ddi_get_iblock_cookie`(9F).

- `SEE ALSO` indicates functions that are related by usage and sources, and which can be referred to for further information.
- `EXAMPLES` shows how the function can be used in driver code.

Every driver MUST include `<sys/ddi.h>` and `<sys/sunddi.h>`, in that order, and as the last files the driver includes.

STREAMS Kernel Function Summary

The following table summarizes the STREAMS functions described in this section.

Routine	Type
`adjmsg`	DDI/DKI
`allocb`	DDI/DKI
`backq`	DDI/DKI
`bcanput`	DDI/DKI
`bcanputnext`	DDI/DKI
`bufcall`	DDI/DKI
`canput`	DDI/DKI
`canputnext`	DDI/DKI
`clrbuf`	DDI/DKI
`copyb`	DDI/DKI
`copymsg`	DDI/DKI
`datamsg`	DDI/DKI
`dupb`	DDI/DKI
`dupmsg`	DDI/DKI
`enableok`	DDI/DKI
`esballoc`	DDI/DKI
`esbbcall`	DDI/DKI
`flushband`	DDI/DKI
`flushq`	DDI/DKI
`freeb`	DDI/DKI
`freemsg`	DDI/DKI
`freezestr`	DDI/DKI
`getq`	DDI/DKI

Routine	Type
insq	DDI/DKI
linkb	DDI/DKI
msgdsize	DDI/DKI
msgpullup	DDI/DKI
mt-streams	Solaris DDI
noenable	DDI/DKI
OTHERQ	DDI/DKI
pullupmsg	DDI/DKI
put	DDI/DKI
putbq	DDI/DKI
putctl	DDI/DKI
putctl1	DDI/DKI
putnext	DDI/DKI
putnextctl	DDI/DKI
putq	DDI/DKI
qbufcall	Solaris DDI
qenable	DDI/DKI
qprocson	DDI/DKI
qprocsoff	DDI/DKI
qreply	DDI/DKI
qsize	DDI/DKI
qtimeout	Solaris DDI
qunbufcall	Solaris DDI
quntimeout	Solaris DDI
qwait	Solaris DDI
qwait_sig	Solaris DDI
qwriter	Solaris DDI
RD	DDI/DKI
rmvb	DDI/DKI

Routine	Type
rmvq	DDI/DKI
SAMESTR	DDI/DKI
strlog	DDI/DKI
strqget	DDI/DKI
strqset	DDI/DKI
testb	DDI/DKI
unbufcall	DDI/DKI
unfreezestr	DDI/DKI
unlinkb	DDI/DKI
WR	DDI/DKI

The following table summarizes the functions not specific to STREAMS.

Routine	Type
ASSERT	DDI/DKI
anocancel	Solaris DDI
aphysio	Solaris DDI
bcmp	DDI/DKI
bcopy	DDI/DKI
biodone	DDI/DKI
bioclone	Solaris DDI
biofini	Solaris DDI
bioinit	Solaris DDI
biomodified	Solaris DDI
biosize	Solaris DDI
bioerror	Solaris DDI
bioreset	Solaris DDI
biowait	DDI/DKI
bp_mapin	DDI/DKI
bp_mapout	DDI/DKI

Routine	Type
btop	DDI/DKI
btopr	DDI/DKI
bzero	DDI/DKI
cmn_err	DDI/DKI
copyin	DDI/DKI
copyout	DDI/DKI
cv_broadcast	Solaris DDI
cv_destroy	Solaris DDI
cv_init	Solaris DDI
cv_signal	Solaris DDI
cv_timedwait	Solaris DDI
cv_wait	Solaris DDI
cv_wait_sig	Solaris DDI
ddi_add_intr	Solaris DDI
ddi_add_softintr	Solaris DDI
ddi_btop	Solaris DDI
ddi_btopr	Solaris DDI
ddi_copyin	Solaris DDI
ddi_copyout	Solaris DDI
ddi_create_minor_node	Solaris DDI
ddi_dev_is_sid	Solaris DDI
ddi_dev_nintrs	Solaris DDI
ddi_dev_nregs	Solaris DDI
ddi_dev_regsize	Solaris DDI
ddi_device_copy	Solaris DDI
ddi_device_zero	Solaris DDI
ddi_devmap_segmap	Solaris DDI
ddi_dma_addr_bind_handle	Solaris DDI
ddi_dma_addr_setup	Solaris DDI

Routine	Type
ddi_dma_alloc_handle	Solaris DDI
ddi_dma_buf_bind_handle	Solaris DDI
ddi_dma_buf_setup	Solaris DDI
ddi_dma_burstsizes	Solaris DDI
ddi_dma_coff	Solaris SPARC DDI
ddi_dma_curwin	Solaris SPARC DDI
ddi_dma_devalign	Solaris DDI
ddi_dma_free	Solaris DDI
ddi_dma_free_handle	Solaris DDI
ddi_dma_getwin	Solaris DDI
ddi_dma_htoc	Solaris SPARC DDI
ddi_dma_mem_alloc	Solaris DDI
ddi_dma_mem_free	Solaris DDI
ddi_dma_movwin	Solaris SPARC DDI
ddi_dma_nextcookie	Solaris DDI
ddi_dma_nextseg	Solaris DDI
ddi_dma_nextwin	Solaris DDI
ddi_dma_numwin	Solaris DDI
ddi_dma_segtocookie	Solaris DDI
ddi_dma_set_sbus64	Solaris DDI
ddi_dma_setup	Solaris DDI
ddi_dma_sync	Solaris DDI
ddi_dma_unbind_handle	Solaris DDI
ddi_dmae	Solaris x86 DDI
ddi_dmae_1stparty	Solaris x86 DDI
ddi_dmae_alloc	Solaris x86 DDI
ddi_dmae_disable	Solaris x86 DDI
ddi_dmae_enable	Solaris x86 DDI
ddi_dmae_getattr	Solaris x86 DDI

Routine	Type
ddi_dmae_getcnt	Solaris x86 DDI
ddi_dmae_getlim	Solaris x86 DDI
ddi_dmae_prog	Solaris x86 DDI
ddi_dmae_release	Solaris x86 DDI
ddi_dmae_stop	Solaris x86 DDI
ddi_enter_critical	Solaris DDI
ddi_exit_critical	Solaris DDI
ddi_ffs	Solaris DDI
ddi_fls	Solaris DDI
ddi_get16	Solaris DDI
ddi_get32	Solaris DDI
ddi_get64	Solaris DDI
ddi_get8	Solaris DDI
ddi_get_cred	Solaris DDI
ddi_get_driver_private	Solaris DDI
ddi_get_iblock_cookie	Solaris DDI
ddi_get_instance	Solaris DDI
ddi_get_name	Solaris DDI
ddi_get_parent	Solaris DDI
ddi_get_soft_iblock_cookie	Solaris DDI
ddi_get_soft_state	Solaris DDI
ddi_getb	Solaris DDI
ddi_getl	Solaris DDI
ddi_getll	Solaris DDI
ddi_getlongprop	Solaris DDI
ddi_getlongprop_buf	Solaris DDI
ddi_getprop	Solaris DDI
ddi_getproplen	Solaris DDI
ddi_getw	Solaris DDI

Routine	Type
ddi_intr_hilevel	Solaris DDI
ddi_io_get16	Solaris DDI
ddi_io_get32	Solaris DDI
ddi_io_get8	Solaris DDI
ddi_io_getb	Solaris DDI
ddi_io_getl	Solaris DDI
ddi_io_getw	Solaris DDI
ddi_io_put16	Solaris DDI
ddi_io_put32	Solaris DDI
ddi_io_put8	Solaris DDI
ddi_io_putb	Solaris DDI
ddi_io_putl	Solaris DDI
ddi_io_putw	Solaris DDI
ddi_io_rep_get16	Solaris DDI
ddi_io_rep_get32	Solaris DDI
ddi_io_rep_get8	Solaris DDI
ddi_io_rep_getb	Solaris DDI
ddi_io_rep_getl	Solaris DDI
ddi_io_rep_getw	Solaris DDI
ddi_io_rep_put16	Solaris DDI
ddi_io_rep_put32	Solaris DDI
ddi_io_rep_put8	Solaris DDI
ddi_io_rep_putb	Solaris DDI
ddi_io_rep_putl	Solaris DDI
ddi_io_rep_putw	Solaris DDI
ddi_iomin	Solaris DDI
ddi_iopb_alloc	Solaris DDI
ddi_iopb_free	Solaris DDI
ddi_map_regs	Solaris DDI

Routine	Type
ddi_mapdev	Solaris DDI
ddi_mapdev_intercept	Solaris DDI
ddi_mapdev_nointercept	Solaris DDI
ddi_mapdev_set_device_acc_attr	Solaris DDI
ddi_mem_alloc	Solaris DDI
ddi_mem_free	Solaris DDI
ddi_mem_get16	Solaris DDI
ddi_mem_get32	Solaris DDI
ddi_mem_get64	Solaris DDI
ddi_mem_get8	Solaris DDI
ddi_mem_getb	Solaris DDI
ddi_mem_getl	Solaris DDI
ddi_mem_getll	Solaris DDI
ddi_mem_getw	Solaris DDI
ddi_mem_put16	Solaris DDI
ddi_mem_put32	Solaris DDI
ddi_mem_put64	Solaris DDI
ddi_mem_put8	Solaris DDI
ddi_mem_putb	Solaris DDI
ddi_mem_putl	Solaris DDI
ddi_mem_putll	Solaris DDI
ddi_mem_putw	Solaris DDI
ddi_mem_rep_get16	Solaris DDI
ddi_mem_rep_get32	Solaris DDI
ddi_mem_rep_get64	Solaris DDI
ddi_mem_rep_get8	Solaris DDI
ddi_mem_rep_getb	Solaris DDI
ddi_mem_rep_getl	Solaris DDI
ddi_mem_rep_getll	Solaris DDI

Routine	Type
ddi_mem_rep_getw	Solaris DDI
ddi_mem_rep_put16	Solaris DDI
ddi_mem_rep_put32	Solaris DDI
ddi_mem_rep_put64	Solaris DDI
ddi_mem_rep_put8	Solaris DDI
ddi_mem_rep_putb	Solaris DDI
ddi_mem_rep_putl	Solaris DDI
ddi_mem_rep_putll	Solaris DDI
ddi_mem_rep_putw	Solaris DDI
ddi_mmap_get_model	Solaris DDI
ddi_model_convert_from	Solaris DDI
ddi_node_name	Solaris DDI
ddi_peek16	Solaris DDI
ddi_peek32	Solaris DDI
ddi_peek64	Solaris DDI
ddi_peek8	Solaris DDI
ddi_peekc	Solaris DDI
ddi_peekd	Solaris DDI
ddi_peekl	Solaris DDI
ddi_peeks	Solaris DDI
ddi_poke16	Solaris DDI
ddi_poke32	Solaris DDI
ddi_poke64	Solaris DDI
ddi_poke8	Solaris DDI
ddi_pokec	Solaris DDI
ddi_poked	Solaris DDI
ddi_pokel	Solaris DDI
ddi_pokes	Solaris DDI
ddi_prop_create	Solaris DDI

Routine	Type
`ddi_prop_exists`	Solaris DDI
`ddi_prop_free`	Solaris DDI
`ddi_prop_get_int`	Solaris DDI
`ddi_prop_lookup`	Solaris DDI
`ddi_prop_lookup_byte_array`	Solaris DDI
`ddi_prop_lookup_int_array`	Solaris DDI
`ddi_prop_lookup_string`	Solaris DDI
`ddi_prop_lookup_string_array`	Solaris DDI
`ddi_prop_modify`	Solaris DDI
`ddi_prop_op`	Solaris DDI
`ddi_prop_remove`	Solaris DDI
`ddi_prop_remove_all`	Solaris DDI
`ddi_prop_undefine`	Solaris DDI
`ddi_prop_update`	Solaris DDI
`ddi_prop_update_byte_array`	Solaris DDI
`ddi_prop_update_int`	Solaris DDI
`ddi_prop_update_int_array`	Solaris DDI
`ddi_prop_update_string`	Solaris DDI
`ddi_prop_update_string_array`	Solaris DDI
`ddi_ptob`	Solaris DDI
`ddi_put16`	Solaris DDI
`ddi_put32`	Solaris DDI
`ddi_put64`	Solaris DDI
`ddi_put8`	Solaris DDI
`ddi_putb`	Solaris DDI
`ddi_putl`	Solaris DDI
`ddi_putll`	Solaris DDI
`ddi_putw`	Solaris DDI
`ddi_regs_map_free`	Solaris DDI

Routine	Type
`ddi_regs_map_setup`	Solaris DDI
`ddi_remove_intr`	Solaris DDI
`ddi_remove_minor_node`	Solaris DDI
`ddi_remove_softintr`	Solaris DDI
`ddi_rep_get16`	Solaris DDI
`ddi_rep_get32`	Solaris DDI
`ddi_rep_get64`	Solaris DDI
`ddi_rep_get8`	Solaris DDI
`ddi_rep_getb`	Solaris DDI
`ddi_rep_getl`	Solaris DDI
`ddi_rep_getll`	Solaris DDI
`ddi_rep_getw`	Solaris DDI
`ddi_rep_put16`	Solaris DDI
`ddi_rep_put32`	Solaris DDI
`ddi_rep_put64`	Solaris DDI
`ddi_rep_put8`	Solaris DDI
`ddi_rep_putb`	Solaris DDI
`ddi_rep_putl`	Solaris DDI
`ddi_rep_putll`	Solaris DDI
`ddi_rep_putw`	Solaris DDI
`ddi_report_dev`	Solaris DDI
`ddi_root_node`	Solaris DDI
`ddi_segmap`	Solaris DDI
`ddi_segmap_setup`	Solaris DDI
`ddi_set_driver_private`	Solaris DDI
`ddi_slaveonly`	Solaris DDI
`ddi_soft_state`	Solaris DDI
`ddi_soft_state_fini`	Solaris DDI
`ddi_soft_state_free`	Solaris DDI

Routine	Type
`ddi_soft_state_init`	Solaris DDI
`ddi_soft_state_zalloc`	Solaris DDI
`ddi_trigger_softintr`	Solaris DDI
`ddi_umem_alloc`	Solaris DDI
`ddi_umem_free`	Solaris DDI
`ddi_unmap_regs`	Solaris DDI
`delay`	DDI/DKI
`devmap_default_access`	Solaris DDI
`devmap_devmem_setup`	Solaris DDI
`devmap_do_ctxmgt`	Solaris DDI
`devmap_load`	Solaris DDI
`devmap_set_ctx_timeout`	Solaris DDI
`devmap_setup`	Solaris DDI
`devmap_umem_setup`	Solaris DDI
`devmap_unload`	Solaris DDI
`disksort`	Solaris DDI
`drv_getparm`	DDI/DKI
`drv_hztousec`	DDI/DKI
`drv_priv`	DDI/DKI
`drv_usectohz`	DDI/DKI
`drv_usecwait`	DDI/DKI
`free_pktiopb`	Solaris DDI
`freerbuf`	DDI/DKI
`get_pktiopb`	Solaris DDI
`geterror`	DDI/DKI
`getmajor`	DDI/DKI
`getminor`	DDI/DKI
`getrbuf`	DDI/DKI
`hat_getkpfnum`	DKI only

Routine	Type
inb	Solaris x86 DDI
inl	Solaris x86 DDI
inw	Solaris x86 DDI
kmem_alloc	DDI/DKI
kmem_free	DDI/DKI
kmem_zalloc	DDI/DKI
kstat_create	Solaris DDI
kstat_delete	Solaris DDI
kstat_install	Solaris DDI
kstat_named_init	Solaris DDI
kstat_queue	Solaris DDI
kstat_runq_back_to_waitq	Solaris DDI
kstat_runq_enter	Solaris DDI
kstat_runq_exit	Solaris DDI
kstat_waitq_enter	Solaris DDI
kstat_waitq_exit	Solaris DDI
kstat_waitq_to_runq	Solaris DDI
makecom_g0	Solaris DDI
makecom_g0_s	Solaris DDI
makecom_g1	Solaris DDI
makecom_g5	Solaris DDI
makedevice	DDI/DKI
max	DDI/DKI
min	DDI/DKI
minphys	Solaris DDI
mod_info	Solaris DDI
mod_install	Solaris DDI
mod_remove	Solaris DDI
mutex_destroy	Solaris DDI

Routine	Type
`mutex_enter`	Solaris DDI
`mutex_exit`	Solaris DDI
`mutex_init`	Solaris DDI
`mutex_owned`	Solaris DDI
`mutex_tryenter`	Solaris DDI
`nochpoll`	Solaris DDI
`nodev`	DDI/DKI
`nulldev`	DDI/DKI
`numtos`	Solaris DDI
`outb`	Solaris x86 DDI
`outl`	Solaris x86 DDI
`outw`	Solaris x86 DDI
`pci_config_get16`	Solaris DDI
`pci_config_get32`	Solaris DDI
`pci_config_get64`	Solaris DDI
`pci_config_get8`	Solaris DDI
`pci_config_getb`	Solaris DDI
`pci_config_getl`	Solaris DDI
`pci_config_getw`	Solaris DDI
`pci_config_put16`	Solaris DDI
`pci_config_put32`	Solaris DDI
`pci_config_put64`	Solaris DDI
`pci_config_put8`	Solaris DDI
`pci_config_putb`	Solaris DDI
`pci_config_putl`	Solaris DDI
`pci_config_putw`	Solaris DDI
`pci_config_setup`	Solaris DDI
`pci_config_teardown`	Solaris DDI
`physio`	Solaris DDI

Routine	Type
pollwakeup	DDI/DKI
proc_ref	Solaris DDI
proc_signal	Solaris DDI
proc_unref	Solaris DDI
ptob	DDI/DKI
repinsb	Solaris x86 DDI
repinsd	Solaris x86 DDI
repinsw	Solaris x86 DDI
repoutsb	Solaris x86 DDI
repoutsd	Solaris x86 DDI
repoutsw	Solaris x86 DDI
rmalloc	DDI/DKI
rmalloc_wait	DDI/DKI
rmallocmap	DDI/DKI
rmallocmap_wait	DDI/DKI
rmfree	DDI/DKI
rmfreemap	DDI/DKI
rw_destroy	Solaris DDI
rw_downgrade	Solaris DDI
rw_enter	Solaris DDI
rw_exit	Solaris DDI
rw_init	Solaris DDI
rw_read_locked	Solaris DDI
rw_tryenter	Solaris DDI
rw_tryupgrade	Solaris DDI
scsi_abort	Solaris DDI
scsi_alloc_consistent_buf	Solaris DDI
scsi_cname	Solaris DDI
scsi_destroy_pkt	Solaris DDI

Routine	Type
scsi_dmafree	Solaris DDI
scsi_dmaget	Solaris DDI
scsi_dname	Solaris DDI
scsi_errmsg	Solaris DDI
scsi_free_consistent_buf	Solaris DDI
scsi_hba_attach	Solaris DDI
scsi_hba_attach_setup	Solaris DDI
scsi_hba_detach	Solaris DDI
scsi_hba_fini	Solaris DDI
scsi_hba_init	Solaris DDI
scsi_hba_lookup_capstr	Solaris DDI
scsi_hba_pkt_alloc	Solaris DDI
scsi_hba_pkt_free	Solaris DDI
scsi_hba_probe	Solaris DDI
scsi_hba_tran_alloc	Solaris DDI
scsi_hba_tran_free	Solaris DDI
scsi_ifgetcap	Solaris DDI
scsi_ifsetcap	Solaris DDI
scsi_init_pkt	Solaris DDI
scsi_log	Solaris DDI
scsi_mname	Solaris DDI
scsi_pktalloc	Solaris DDI
scsi_pktfree	Solaris DDI
scsi_poll	Solaris DDI
scsi_probe	Solaris DDI
scsi_resalloc	Solaris DDI
scsi_reset	Solaris DDI
scsi_reset_notify	Solaris DDI
scsi_resfree	Solaris DDI

Routine	Type
`scsi_rname`	Solaris DDI
`scsi_slave`	Solaris DDI
`scsi_sname`	Solaris DDI
`scsi_sync_pkt`	Solaris DDI
`scsi_transport`	Solaris DDI
`scsi_unprobe`	Solaris DDI
`scsi_unslave`	Solaris DDI
`sema_destroy`	Solaris DDI
`sema_init`	Solaris DDI
`sema_p`	Solaris DDI
`sema_p_sig`	Solaris DDI
`sema_tryp`	Solaris DDI
`sema_v`	Solaris DDI
`sprintf`	Solaris DDI
`stoi`	Solaris DDI
`strchr`	Solaris DDI
`strcmp`	Solaris DDI
`strcpy`	Solaris DDI
`strlen`	Solaris DDI
`strncmp`	Solaris DDI
`strncpy`	Solaris DDI
`swab`	DDI/DKI
`timeout`	DDI/DKI
`uiomove`	DDI/DKI
`untimeout`	DDI/DKI
`ureadc`	DDI/DKI
`uwritec`	DDI/DKI
`va_arg`	Solaris DDI
`va_end`	Solaris DDI

Routine	Type
`va_start`	Solaris DDI
`vcmn_err`	DDI/DKI
`vsprintf`	Solaris DDI

Kernel Functions for Drivers

NAME

copyb – copy a message block

SYNOPSIS

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

mblk_t *copyb(mblk_t *bp);
```

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

PARAMETERS

bp	Pointer to the message block from which data is copied.

DESCRIPTION

`copyb()` allocates a new message block, and copies into it the data from the block that *bp* denotes. The new block will be at least as large as the block being copied. `copyb()` uses the `b_rptr` and `b_wptr` members of *bp* to determine how many bytes to copy.

RETURN VALUES

If successful, `copyb()` returns a pointer to the newly allocated message block containing the copied data. Otherwise, it returns a `NULL` pointer.

CONTEXT

`copyb()` can be called from user or interrupt context.

EXAMPLES

EXAMPLE 1 Using copyb

For each message in the list, test to see if the downstream queue is full with the `canputnext`(9F) function (line 21). If it is not full, use `copyb()` to copy a header message block, and `dupmsg`(9F) to duplicate the data to be retransmitted. If either operation fails, reschedule a timeout at the next valid interval.

Update the new header block with the correct destination address (line 34), link the message to it (line 35), and send it downstream (line 36). At the end of the list, reschedule this routine.

```
 1  struct retrans {
 2        mblk_t             *r_mp;
 3        int                r_address;
 4        queue_t            *r_outq;
 5        struct retrans     *r_next;
 6  };
 7
 8  struct protoheader {
       ...
 9     int                    h_address;
       ...
10  };
11
12  mblk_t *header;
13
14  void
15  retransmit(struct retrans *ret)
16  {
17       mblk_t *bp, *mp;
18       struct protoheader *php;
19
20       while (ret) {
21         if (!canputnext(ret->r_outq)) {   /* no room */
```

EXAMPLE 1 Using copyb *(Continued)*

```
              ret = ret->r_next;
              continue;
       }
       bp = copyb(header);               /* copy header msg. block */
       if (bp == NULL)
             break;
       mp = dupmsg(ret->r_mp);           /* duplicate data */
       if (mp == NULL) {                 /* if unsuccessful */
            freeb(bp);                   /* free the block */
            break;
       }
       php = (struct protoheader *)bp->b_rptr;
       php->h_address = ret->r_address;   /* new header */
       bp->bp_cont = mp;                  /* link the message */
       putnext(ret->r_outq, bp);          /* send downstream */
       ret = ret->r_next;
     }
     /* reschedule */
     (void) timeout(retransmit, (caddr_t)ret, RETRANS_TIME);
}
```

SUMMARY OF TRUSTED SOLARIS CHANGES This routine will assign correct attributes to the message it copies.

Trusted Solaris 8 4/01 Reference Manual dupmsg(9F)

SunOS 5.8 Reference Manual attributes(5), allocb(9F), canputnext(9F)

Writing Device Drivers

STREAMS Programming Guide

NOTES These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

copymsg – Copy a message

SYNOPSIS

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

mblk_t *copymsg(mblk_t *mp);
```

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

PARAMETERS

mp	Pointer to the message to be copied.

DESCRIPTION

`copymsg()` forms a new message by allocating new message blocks and copying the contents of the message referred to by *mp*, using the `copyb`(9F) function. It returns a pointer to the new message.

RETURN VALUES

If the copy is successful, `copymsg()` returns a pointer to the new message. Otherwise, it returns a `NULL` pointer.

CONTEXT

`copymsg()` can be called from user or interrupt context.

EXAMPLES

EXAMPLE 1 Using copymsg

The routine `lctouc()` converts all the lowercase ASCII characters in the message to uppercase. If the reference count is greater than one (line 8), then the message is shared, and must be copied before changing the contents of the data buffer. If the call to the `copymsg()` function fails (line 9), return `NULL` (line 10), otherwise, free the original message (line 11). If the reference count is equal to `1`, the message can be modified. For each character (line 16) in each message block (line 15), if it is a lowercase letter, convert it to an uppercase letter (line 18). A pointer to the converted message is returned (line 21).

```
 1 mblk_t *lctouc(mp)
 2    mblk_t *mp;
 3 {
 4    mblk_t *cmp;
 5    mblk_t *tmp;
 6    unsigned char *cp;
 7
 8    if (mp->b_datap->db_ref > 1) {
 9            if ((cmp = copymsg(mp)) == NULL)
10                     return (NULL);
11            freemsg(mp);
12    } else {
13            cmp = mp;
14    }
15    for (tmp = cmp; tmp; tmp = tmp->b_cont) {
16            for (cp = tmp->b_rptr; cp < tmp->b_wptr; cp++) {
17                    if ((*cp <= 'z') && (*cp >= 'a'))
18                            *cp -= 0x20;
19            }
20    }
21    return(cmp);
22 }
```

EXAMPLE 1 Using copymsg *(Continued)*

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will assign correct attributes to the message it copies.

Trusted Solaris 8 4/01 Reference Manual

`copyb`(9F)

SunOS 5.8 Reference Manual

`allocb`(9F), `msgb`(9S)

Writing Device Drivers

STREAMS Programming Guide

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

dupb – Duplicate a message block descriptor

SYNOPSIS

`#include <sys/stream.h>`

`mblk_t *`**`dupb`**`(mblk_t *`*`bp`*`);`

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

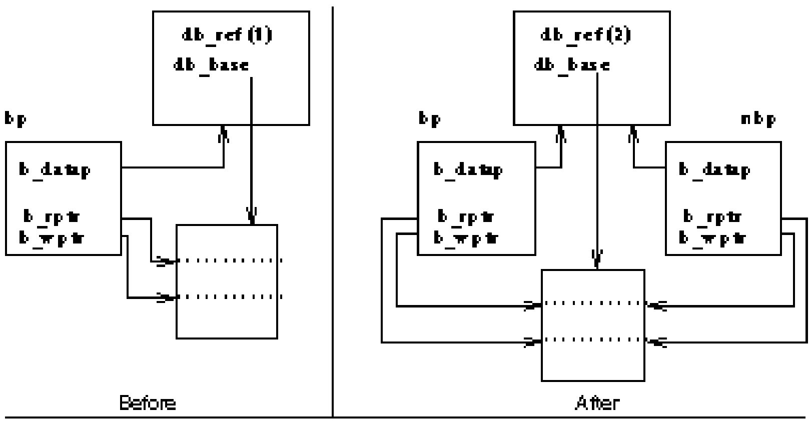
PARAMETERS

bp	Pointer to the message block to be duplicated. `mblk_t` is an instance of the `msgb`(9S) structure.

DESCRIPTION

`dupb()` creates a new `mblk_t` structure (see `msgb`(9S)) to reference the message block pointed to by *bp*.

Unlike `copyb`(9F), `dupb()` does not copy the information in the `dblk_t` structure (see `datab`(9S)), but creates a new `mblk_t` structure to point to it. The reference count in the `dblk_t` structure (`db_ref`) is incremented. The new `mblk_t` structure contains the same information as the original. Note that `b_rptr` and `b_wptr` are copied from the *bp*.



RETURN VALUES

If successful, `dupb()` returns a pointer to the new message block. A `NULL` pointer is returned if `dupb()` cannot allocate a new message block descriptor or if the `db_ref` field of the data block structure (see `datab`(9S)) has reached a maximum value (`255`).

CONTEXT

`dupb()` can be called from user, kernel, or interrupt context.

EXAMPLES

EXAMPLE 1 Using `dupb()`

This `srv`(9E) (service) routine adds a header to all `M_DATA` messages before passing them along. `dupb()` is used instead of `copyb`(9F) because the contents of the header block are not changed.

EXAMPLE 1 Using `dupb()` *(Continued)*

For each message on the queue, if it is a priority message, pass it along immediately (lines 10–11). Otherwise, if it is anything other than an `M_DATA` message (line 12), and if it can be sent along (line 13), then do so (line 14). Otherwise, put the message back on the queue and return (lines 16–17). For all `M_DATA` messages, first check to see if the stream is flow-controlled (line 20). If it is, put the message back on the queue and return (lines 37–38). If it is not, the header block is duplicated (line 21).

`dupb()` can fail either due to lack of resources or because the message block has already been duplicated 255 times. In order to handle the latter case, the example calls `copyb`(9F) (line 22). If `copyb`(9F) fails, it is due to buffer allocation failure. In this case, `qbufcall`(9F) is used to initiate a callback (lines 30-31) if one is not already pending (lines 26-27).

The callback function, `xxxcallback()`, clears the recorded `qbufcall`(9F) callback id and schedules the service procedure (lines 49-50). Note that the close routine, `xxxclose()`, must cancel any outstanding `qbufcall`(9F) callback requests (lines 58-59).

If `dupb()` or `copyb`(9F) succeed, link the `M_DATA` message to the new message block (line 34) and pass it along (line 35).

```
 1  xxxsrv(q)
 2     queue_t *q;
 3  {
 4  struct xx *xx = (struct xx *)q->q_ptr;
 5  mblk_t *mp;
 6  mblk_t *bp;
 7  extern mblk_t *hdr;
 8
 9  while ((mp = getq(q)) != NULL) {
10       if (mp->b_datap->db_type >= QPCTL) {
11            putnext(q, mp);
12       } else if (mp->b_datap->db_type != M_DATA) {
13            if (canputnext(q))
14                 putnext(q, mp);
15            else {
16                 putbq(q, mp);
17                 return;
18            }
19       } else {  /* M_DATA */
20            if (canputnext(q)) {
21                 if ((bp = dupb(hdr)) == NULL)
22                      bp = copyb(hdr);
23                 if (bp == NULL) {
24                      size_t size = msgdsize(mp);
25                      putbq(q, mp);
26                      if (xx->xx_qbufcall_id) {
27                           /* qbufcall pending */
28                           return;
29                      }
30                      xx->xx_qbufcall_id = qbufcall(q, size,
```

EXAMPLE 1 Using `dupb()` *(Continued)*

```
                          BPRI_MED, xxxcallback, (intptr_t)q);
                     return;
                }
                linkb(bp, mp);
                putnext(q, bp);
           } else {
                putbq(q, mp);
                return;
           }
      }
 }
}
 void
 xxxcallback(q)
      queue_t *q;
 {
      struct xx *xx = (struct xx *)q->q_ptr;

      xx->xx_qbufcall_id = 0;
      qenable(q);
 }

 xxxclose(q, cflag, crp)
      queue_t *q;
      int  cflag;
      cred_t *crp;
 {
      struct xx *xx = (struct xx *)q->q_ptr;
      ...
      if (xx->xx_qbufcall_id)
           qunbufcall(q, xx->xx_qbufcall_id);
      ...
 }
```

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will assign correct attributes to the message it copies.

Trusted Solaris 8 4/01 Reference Manual

`copyb`(9F)

SunOS 5.8 Reference Manual

`srv`(9E), `allocb`(9F), `qbufcall`(9F), `datab`(9S), `msgb`(9S)

Writing Device Drivers

STREAMS Programming Guide

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

dupmsg – Duplicate a message

SYNOPSIS

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

mblk_t *dupmsg(mblk_t *mp);
```

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

PARAMETERS

mp	Pointer to the message.

DESCRIPTION

`dupmsg()` forms a new message by copying the message block descriptors pointed to by *mp* and linking them. `dupb`(9F) is called for each message block. The data blocks themselves are not duplicated.

RETURN VALUES

If successful, `dupmsg()` returns a pointer to the new message block. Otherwise, it returns a `NULL` pointer. A return value of `NULL` indicates either memory depletion or the data block reference count, `db_ref` (see `datab`(9S)), has reached a limit (`255`). See `dupb`(9F).

CONTEXT

`dupmsg()` can be called from user, kernel, or interrupt context.

EXAMPLES

EXAMPLE 1 Using dupmsg

See `copyb`(9F) for an example using `dupmsg()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will preserve the attributes of the message it manipulates.

Trusted Solaris 8 4/01 Reference Manual

`copyb`(9F), `copymsg`(9F), `dupb`(9F)

SunOS 5.8 Reference Manual

`datab`(9S)

Writing Device Drivers

STREAMS Programming Guide

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

insq – Insert a message into a queue

SYNOPSIS

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

int insq(queue_t *q, mblk_t *emp, mblk_t *nmp);
```

PARAMETERS

q	Pointer to the queue containing message *emp*.
emp	Enqueued message before which the new message is to be inserted. `mblk_t` is an instance of the `msgb`(9S) structure.
nmp	Message to be inserted.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`insq()` inserts a message into a queue. The message to be inserted*, nmp,* is placed in *q* immediately before the message *emp*. If *emp* is `NULL`, the new message is placed at the end of the queue. The queue class of the new message is ignored. All flow control parameters are updated. The service procedure is enabled unless `QNOENB` is set.

RETURN VALUES

`insq()` returns:

`1`	On success.
`0`	On failure.

CONTEXT

`insq()` can be called from user or interrupt context.

EXAMPLES

This routine illustrates the steps a transport provider may take to place expedited data ahead of normal data on a queue (assume all `M_DATA` messages are converted into `M_PROTO T_DATA_REQ` messages). Normal `T_DATA_REQ` messages are just placed on the end of the queue (line 16). However, expedited `T_EXDATA_REQ` messages are inserted before any normal messages already on the queue (line 25). If there are no normal messages on the queue, *bp* will be `NULL` and we fall out of the `for` loop (line 21). `insq()` acts like `putq`(9F) in this case.

```
 1  #include <sys/tihdr.h>
 2  #include <sys/stream.h>
 3
 4  static int
 5  xxxwput(queue_t *q, mblk_t *mp)
 6  {
 7   union T_primitives *tp;
 8   mblk_t *bp;
 9   union T_primitives *ntp;
10
11   switch (mp->b_datap->db_type) {
12   case M_PROTO:
13        tp = (union T_primitives *)mp->b_rptr;
14        switch (tp->type) {
15        case T_DATA_REQ:
16                putq(q, mp);
17                break;
18
19        case T_EXDATA_REQ:
```

```
              freezestr(q);
              for (bp = q->q_first; bp; bp = bp->b_next) {
                 if (bp->b_datap->db_type == M_PROTO) {
                   ntp = (union T_primitives *)bp->b_rptr;
                   if (ntp->type != T_EXDATA_REQ)
                       break;
                }
              }
              (void)insq(q, bp, mp);
              unfreezestr(q);
              break;
      . . .
            }
 }
}
```

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will try to assign attribute structures to the `mblks` of a message that does not have one. The first attribute stucture found will be used. If a stream module illegally combined messages, `mblks` can have different attribute structures; in that case, the message will be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`Intro`(9F), `putq`(9F)

SunOS 5.8 Reference Manual

`freezestr`(9F), `rmvq`(9F), `unfreezestr`(9F), `msgb`(9S)

Writing Device Drivers

STREAMS Programming Guide

WARNINGS

If *emp* is non-NULL, it must point to a message on *q* or a system panic could result.

NOTES

The stream must be frozen using `freezestr`(9F) before calling `insq()`.

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

kstat_create – Create and initialize a new kstat

SYNOPSIS

```
#include <sys/types.h>
#include <sys/kstat.h>
```

```
kstat_t *kstat_create(char *module, int instance, char *name, char
     *class, uchar_t type, ulong_t ndata, uchar_t ks_flag);
```

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI)

PARAMETERS

module The name of the provider's module (such as "sd", "esp", ...). The "core" kernel uses the name "unix".

instance The provider's instance number, as from `ddi_get_instance`(9F). Modules which do not have a meaningful instance number should use `0`.

name A pointer to a string that uniquely identifies this structure. Only `KSTAT_STRLEN – 1` characters are significant.

class The general class that this kstat belongs to. The following classes are currently in use: `disk`, `tape`, `net`, `controller`, `vm`, `kvm`, `hat`, `streams`, `kstat`, and `misc`.

type The type of `kstat` to allocate. Valid types are:

`KSTAT_TYPE_NAMED`
Allows more than one data record per `kstat`.

`KSTAT_TYPE_INTR`
Interrupt; only one data record per `kstat`.

`KSTAT_TYPE_IO`
I/O; only one data record per `kstat`

ndata The number of type-specific data records to allocate.

flag A bit-field of various flags for this `kstat`. *flag* is some combination of:

`KSTAT_FLAG_VIRTUAL`
Tells `kstat_create()` not to allocate memory for the `kstat` data section; instead, the driver will set the `ks_data` field to point to the data it wishes to export. This provides a convenient way to export existing data structures.

`KSTAT_FLAG_WRITABLE`
Makes the kstat data section writable by a process with the `PRIV_SYS_CONFIG` privilege and MAC write access to `/dev/kstat`.

`KSTAT_FLAG_PERSISTENT`
Indicates that this `kstat` is to be persistent over time. For persistent `kstats`, `kstat_delete`(9F) simply marks the

kstat as dormant; a subsequent kstat_create() reactivates the kstat. This feature is provided so that statistics are not lost across driver close/open (such as raw disk I/O on a disk with no mounted partitions.) Note: Persistent kstats cannot be virtual, since ks_data points to garbage as soon as the driver goes away.

DESCRIPTION

kstat_create() is used in conjunction with kstat_install(9F) to allocate and initialize a kstat(9S) structure. The method is generally as follows:

kstat_create() allocates and performs necessary system initialization of a kstat(9S) structure. kstat_create() allocates memory for the entire kstat (header plus data), initializes all header fields, initializes the data section to all zeroes, assigns a unique kstat ID (KID), and puts the kstat onto the system's kstat chain. The returned kstat is marked invalid because the provider (caller) has not yet had a chance to initialize the data section.

After a successful call to kstat_create(), the driver must perform any necessary initialization of the data section (such as setting the name fields in a kstat of type KSTAT_TYPE_NAMED). Virtual kstats must have the ks_data field set at this time. The provider may also set the ks_update, ks_private, and ks_lock fields if necessary.

Once the kstat is completely initialized, kstat_install(9F) is used to make the kstat accessible to the outside world.

RETURN VALUES

If successful, kstat_create() returns a pointer to the allocated kstat. NULL is returned upon failure.

CONTEXT

kstat_create() can be called from user or kernel context.

EXAMPLES

EXAMPLE 1 Allocating and Initializing a kstat Structure

```
pkstat_t    *ksp;
    ksp = kstat_create(module, instance, name, class, type, ndata, flags);
    if (ksp) {
       /* ... provider initialization, if necessary */
       kstat_install(ksp);
    }
```

SUMMARY OF TRUSTED SOLARIS CHANGES

The KSTAT_FLAG_WRITABLE flag makes the kstat data section writable by a process with the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.

Trusted Solaris 8 4/01 Reference Manual

Intro(9F)

SunOS 5.8 Reference Manual

kstat(3KSTAT), ddi_get_instance(9F), attributes(5), kstat_delete(9F), kstat_install(9F), kstat_named_init(9F), kstat(9S), kstat_named(9S)

Writing Device Drivers

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

linkb – concatenate two message blocks

SYNOPSIS

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

void **linkb**(mblk_t **mp1*, mblk_t **mp2*);

PARAMETERS

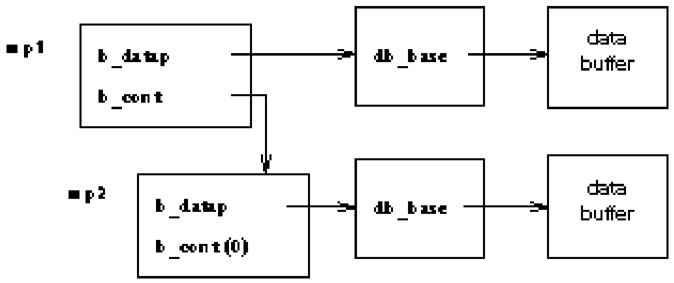
mp1	The message to which *mp2* is to be added. `mblk_t` is an instance of the `msgb`(9S) structure.
mp2	The message to be added.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`linkb()` creates a new message by adding *mp2* to the tail of *mp1*. The continuation pointer, `b_cont`, of *mp1* is set to point to *mp2*.



linkb(mp1, mp2);

CONTEXT

`linkb()` can be called from user or interrupt context.

EXAMPLES

See `dupb`(9F) for an example of using `linkb()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

Module writers for the Trusted Solaris environment should check that the attributes of *mp1* and *mp2* are equivalent before calling this function. Mixing messages with non-equivalent attributes can result in a violation of the security policy.

Trusted Solaris 8 4/01 Reference Manual

`dupb`(9F), `unlinkb`(9F)

SunOS 5.8 Reference Manual

`msgb`(9S)

Writing Device Drivers

STREAMS Programming Guide

NAME

msgpullup – Concatenate bytes in a message

SYNOPSIS

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

```
mblk_t *msgpullup(mblk_t *mp, ssize_t len);
```

PARAMETERS

mp	Pointer to the message whose blocks are to be concatenated.
len	Number of bytes to concatenate.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`msgpullup()` concatenates and aligns the first *len* data bytes of the message pointed to by *mp*, copying the data into a new message. Any remaining bytes in the remaining message blocks will be copied and linked onto the new message. The original message is unaltered. If *len* equals `−1`, all data are concatenated. If *len* bytes of the same message type cannot be found, `msgpullup()` fails and returns `NULL`.

RETURN VALUES

Upon successful completion, `msgpullup()` returns a pointer to the new message. Upon failure, `msgpullup()` returns `NULL`.

CONTEXT

`msgpullup()` can be called from user or interrupt context.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will try to assign attribute structures to the `mblks` of amessage that does not have one. The first attribute stucture found will be used. If a stream module illegally combined messages, `mblks` can have different attribute structures; in that case, the message will be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`intro`(9F), `pullupmsg`(9F)

SunOS 5.8 Reference Manual

`srv`(9E) , `allocb`(9F), `msgb`(9S)

Writing Device Drivers

STREAMS Programming Guide

NOTES

`msgpullup()` is a DKI-compliant replacement for the older `pullupmsg`(9F) routine. Users are strongly encouraged to use `msgpullup()` instead of `pullupmsg`(9F).

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

pullupmsg – Concatenate bytes in a message

SYNOPSIS

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

int pullupmsg(mblk_t *mp, ssize_t len);
```

PARAMETERS

mp	Pointer to the message whose blocks are to be concatenated. `mblk_t` is an instance of the `msgb`(9S) structure.
len	Number of bytes to concatenate.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`pullupmsg()` tries to combine multiple data blocks into a single block. `pullupmsg()` concatenates and aligns the first *len* data bytes of the message pointed to by *mp*. If *len* equals `-1`, all data are concatenated. If *len* bytes of the same message type cannot be found, `pullupmsg()` fails and returns `0`.

RETURN VALUES

`pullupmsg()` returns:

`1`	On success.
`0`	On failure.

CONTEXT

`pullupmsg()` can be called from user or interrupt context.

EXAMPLES

EXAMPLE 1 Using pullupmsg

This is a driver write `srv`(9E) (service) routine for a device that does not support scatter/gather DMA. For all `M_DATA` messages, the data will be transferred to the device with DMA.

First, try to pull up the message into one message block with the `pullupmsg()` function (line 12). If successful, the transfer can be accomplished in one DMA job. Otherwise, it must be done one message block at a time (lines 19–22). After the data has been transferred to the device, free the message and continue processing messages on the queue.

```
 1 xxxwsrv(q)
 2    queue_t *q;
 3 {
 4        mblk_t *mp;
 5        mblk_t *tmp;
 6        caddr_t dma_addr;
 7     ssize_t dma_len;
 8
 9        while ((mp = getq(q)) != NULL) {
10                switch (mp->b_datap->db_type) {
11             case M_DATA:
12                       if (pullupmsg(mp, -1)) {
13                            dma_addr = vtop(mp->b_rptr);
14                               dma_len = mp->b_wptr - mp->b_rptr;
15                               xxx_do_dma(dma_addr, dma_len);
16                               freemsg(mp);
```

EXAMPLE 1 Using pullupmsg *(Continued)*

```
                              break;
                      }
                      for (tmp = mp; tmp; tmp = tmp->b_cont) {
                           dma_addr = vtop(tmp->b_rptr);
                           dma_len = tmp->b_wptr - tmp->b_rptr;
                           xxx_do_dma(dma_addr, dma_len);
                   }
                      freemsg(mp);
                      break;
       . . .
               }
       }
}
```

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will try to assign attribute structures to the `mblks` of a message that does not have one. The first attribute stucture found will be used. If a stream module illegally combined messages, `mblks` can have different attribute structures; in that case, the message will be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`Intro`(9F), `msgpullup`(9F)

SunOS 5.8 Reference Manual

`allocb`(9F), `msgb`(9S), `srv`(9E)

Writing Device Drivers

STREAMS Programming Guide

NOTES

`pullupmsg()` is not included in the DKI and will be removed from the system in a future release. Device driver writers are strongly encouraged to use `msgpullup`(9F) instead of `pullupmsg()`.

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

put – Call a STREAMS `put()` procedure

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/ddi.h>

void put(queue_t *q, mblk_t *mp);
```

PARAMETERS

q	Pointer to a STREAMS queue.
mp	Pointer to message block being passed into queue.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`put()` calls the put procedure (`put`(9E) entry point) for the STREAMS queue specified by *q*, passing it the message block referred to by *mp*. It is typically used by a driver or module to call its own `put()` procedure.

CONTEXT

`put()` can be called from a STREAMS module or driver `put()` or service routine, or from an associated interrupt handler, timeout, bufcall, or esballoc call-back. In the latter cases the calling code must guarantee the validity of the *q* argument.

Since `put()` may cause re-entry of the module (as it is intended to do), mutexes or other locks should not be held across calls to it, due to the risk of single-party deadlock (`put`(9E), `putnext`(9F), `putctl`(9F), `qreply`(9F)). This function is provided as a DDI/DKI conforming replacement for a direct call to a `put()` procedure.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will try to assign attribute structures to the `mblk`s of a message that does not have one. The first attribute stucture found will be used. If a stream module illegally combined messages, `mblk`s can have different attribute structures; in that case, the message will be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`putctl`(9F), `putctl1`(9F), `putnext`(9F), `putnextctl`(9F), `putnextctl1`(9F)

SunOS 5.8 Reference Manual

`put`(9E), `freezestr`(9F), `qreply`(9F)

Writing Device Drivers

STREAMS Programming Guide

NOTES

The caller cannot have the stream frozen when calling this function. See `freezestr`(9F).

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

DDI/DKI conforming modules and drivers are no longer permitted to call `put()` procedures directly, but must call through the appropriate STREAMS utility function, for example, `put`(9E), `putnext`(9F), `putctl`(9F), and `qreply`(9F). This function is provided as a DDI/DKI conforming replacement for a direct call to a `put()` procedure.

NAME

putbq – Place a message at the head of a queue

SYNOPSIS

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

`int` **`putbq`**`(queue_t *`*q*`, mblk_t *`*bp*`);`

PARAMETERS

q	Pointer to the queue.
bp	Pointer to the message block.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putbq()` places a message at the beginning of the appropriate section of the message queue. There are always sections for high priority and ordinary messages. If other priority bands are used, each will have its own section of the queue, in priority band order, after high priority messages and before ordinary messages. `putbq()` can be used for ordinary, priority band, and high priority messages. However, unless precautions are taken, using `putbq()` with a high priority message is likely to lead to an infinite loop of putting the message back on the queue, being rescheduled, pulling it off, and putting it back on.

This function is usually called when `bcanput`(9F) or `canput`(9F) determines that the message cannot be passed on to the next stream component. The flow control parameters are updated to reflect the change in the queue's status. If `QNOENB` is not set, the service routine is enabled.

RETURN VALUES

`audit()` returns:

`1`	On success.
`0`	On failure.

CONTEXT

`putbq()` can be called from user or interrupt context.

EXAMPLES

See the `bufcall`(9F) function page for an example of `putbq()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine tries to assign attribute structures to the `mblks` of a message that does not have any. The first attribute structure found is used. If a STREAMS module illegally combined messages, `mblks` can have different attribute structures; in that case, the message might be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`putq`(9F)

SunOS 5.8 Reference Manual

`bcanput`(9F), `bufcall`(9F), `canput`(9F), `getq`(9F)

Writing Device Drivers

STREAMS Programming Guide

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

putctl1 – send a control message with a one-byte parameter to a queue

SYNOPSIS

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

int putctl1(queue_t *q, int type, int p);
```

PARAMETERS

q	Queue to which the message is to be sent.
type	Type of message.
p	One-byte parameter.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putctl1()`, like `putctl`(9F), tests the *type* argument to make sure a data type has not been specified, and attempts to allocate a message block. The *p* parameter can be used, for example, to specify how long the delay will be when an `M_DELAY` message is being sent. `putctl1()` fails if *type* is `M_DATA`, `M_PROTO`, or `M_PCPROTO`, or if a message block cannot be allocated. If successful, `putctl1()` calls the `put`(9E) routine of the queue pointed to by *q* with the newly allocated and initialized message.

RETURN VALUES

On success, `1` is returned. `0` is returned if *type* is a data type or if a message block cannot be allocated.

CONTEXT

`putctl1()` can be called from user or interrupt context.

EXAMPLES

See the `putctl`(9F) function page for an example of `putctl1()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

The new message allocated by this routine is labeled with the default streams attributes of the target queue.

Trusted Solaris 8 4/01 Reference Manual

`putctl`(9F), `putnextctl1`(9F)

SunOS 5.8 Reference Manual

`put`(9E), `allocb`(9F), `datamsg`(9F)

Writing Device Drivers

STREAMS Programming Guide

NAME

putctl – send a control message to a queue

SYNOPSIS

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

int putctl(queue_t *q, int type);
```

PARAMETERS

q	Queue to which the message is to be sent.
type	Message type (must be control, not data type).

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putctl()` tests the *type* argument to make sure a data type has not been specified, and then attempts to allocate a message block. `putctl()` fails if *type* is `M_DATA`, `M_PROTO`, or `M_PCPROTO`, or if a message block cannot be allocated. If successful, `putctl()` calls the `put`(9E) routine of the queue pointed to by *q* with the newly allocated and initialized messages.

RETURN VALUES

On success, `1` is returned. If *type* is a data type, or if a message block cannot be allocated, `0` is returned.

CONTEXT

`putctl()` can be called from user or interrupt context.

EXAMPLES

EXAMPLE 1 Using putctl

The `send_ctl()` routine is used to pass control messages downstream. `M_BREAK` messages are handled with `putctl()` (line 11). `putctl1`(9F) (line 16) is used for `M_DELAY` messages, so that *parm* can be used to specify the length of the delay. In either case, if a message block cannot be allocated a variable recording the number of allocation failures is incremented (lines 12, 17). If an invalid message type is detected, `cmn_err`(9F) panics the system (line 21).

```
 1  void
 2  send_ctl(wrq, type, parm)
 3      queue_t *wrq;
 4      uchar_t type;
 5      uchar_t parm;
 6  {
 7          extern int num_alloc_fail;
 8
 9       switch (type) {
10          case M_BREAK:
11             if (!putctl(wrq->q_next, M_BREAK))
12                      num_alloc_fail++;
13                break;
14
15       case M_DELAY:
16             if (!putctl1(wrq->q_next, M_DELAY, parm))
17                      num_alloc_fail++;
18             break;
19
20       default:
21             cmn_err(CE_PANIC, "send_ctl: bad message type passed");
22                break;
```

EXAMPLE 1 Using putctl *(Continued)*

```
        }
}
```

SUMMARY OF TRUSTED SOLARIS CHANGES

The new message allocated by this routine is labeled with the default streams attributes of the target queue.

Trusted Solaris 8 4/01 Reference Manual

putctl(9F), putctl1(9F), putnextctl(9F), putnextctl1(9F)

SunOS 5.8 Reference Manual

cmn_err(9F), datamsg(9F), put(9E)

Writing Device Drivers

STREAMS Programming Guide

NAME

putnext – Send a message to the next queue

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/ddi.h>

void putnext(queue_t *q, mblk_t *mp);
```

PARAMETERS

q	Pointer to the queue from which the message *mp* will be sent.
mp	Message to be passed.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putnext()` is used to pass a message to the `put`(9E) routine of the next queue in the stream.

RETURN VALUES

None.

CONTEXT

`putnext()` can be called from user or interrupt context.

EXAMPLES

See `allocb`(9F) for an example of using `putnext()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine will try to assign attribute structures to the `mblks` of a message that does not have one. The first attribute structure found will be used. If a stream module illegally combined messages, `mblks` can have different attribute structures; in that case, the message will be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`Intro`(9F)

SunOS 5.8 Reference Manual

`allocb`(9F), `put`(9E)

Writing Device Drivers

STREAMS Programming Guide

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

putnextctl1 – send a control message with a one-byte parameter to a queue

SYNOPSIS

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

int putnextctl1(queue_t *q, int type, int p);
```

PARAMETERS

q	Queue to which the message is to be sent.
type	Type of message.
p	One-byte parameter.

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putnextctl1()`, like `putctl1`(9F), tests the *type* argument to make sure a data type has not been specified, and attempts to allocate a message block. The*p* parameter can be used, for example, to specify how long the delay will be when an `M_DELAY` message is being sent. `putnextctl1()` fails if *type* is `M_DATA`, `M_PROTO`, or `M_PCPROTO`, or if a message block cannot be allocated. If successful, `putnextctl1()` calls the `put`(9E) routine of the queue pointed to by *q* with the newly allocated and initialized message.

A call to `putnextctl1(`*q*,*type*, *p*`)` is an atomic equivalent of `putctl1(`*q*`->q_next,` *type*, *p*`)`. The STREAMS framework provides whatever mutual exclusion is necessary to insure that de-referencing *q* through its `q_next` field and then invoking `putctl1`(9F) proceeds without interference from other threads.

`putnextctl1`(9F) `putctl1`(9F).

RETURN VALUES

On success, `1` is returned. `0` is returned if *type* is a data type, or if a message block cannot be allocated.

CONTEXT

`putnextctl1()` can be called from user or interrupt context.

EXAMPLES

See the `putnextctl`(9F) function page for an example of `putnextctl1()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

The new message allocated by this routine is labeled with the default streams attributes of the target queue.

Trusted Solaris 8 4/01 Reference Manual

`putctl`(9F), `putctl1`(9F), `putnextctl1`(9F)

SunOS 5.8 Reference Manual

`allocb`(9F), `put`(9E), `datamsg`(9F)

Writing Device Drivers, *STREAMS Programming Guide*

NAME

putnextctl – send a control message to a queue

SYNOPSIS

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

int putnextctl(queue_t *q, int type);
```

PARAMETERS

q	Queue to which the message is to be sent.
type	Message type (must be control, not data type).

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

DESCRIPTION

`putnextctl()` tests the *type* argument to make sure a data type has not been specified, and then attempts to allocate a message block. `putnextctl()` fails if *type* is `M_DATA`, `M_PROTO`, or `M_PCPROTO`, or if a message block cannot be allocated. If successful, `putnextctl()` calls the `put`(9E) routine of the queue pointed to by *q* with the newly allocated and initialized messages.

A call to `putnextctl(`*q*`,`*type*`)` is an atomic equivalent of `putctl(`*q->q_next*`,`*type*`)`. The STREAMS framework provides whatever mutual exclusion is necessary to ensure that dereferencing *q* through its `q_next` field and then invoking `putctl`(9F) proceeds without interference from other threads.

`putnextctl()` should always be used in preference to `putctl`(9F).

RETURN VALUES

On success, `1` is returned. If *type* is a data type, or if a message block cannot be allocated, `0` is returned.

CONTEXT

`putnextctl()` can be called from user or interrupt context.

EXAMPLES

EXAMPLE 1

The `send_ctl()` routine is used to pass control messages downstream. messages are handled with `putnextctl()` (line 8). `putnextctl1`(9F) (line 13) is used for `M_DELAY` messages, so that *parm* can be used to specify the length of the delay. In either case, if a message block cannot be allocated, a variable recording the number of allocation failures is incremented (lines 9, 14). If an invalid message type is detected, `cmn_err`(9F) panics the system (line 18).

```
 1  void
 2  send_ctl(queue_t *wrq, uchar_t type, uchar_t parm)
 3  {
 4                 extern int num_alloc_fail;
 5
 6                 switch (type) {
 7              case M_BREAK:
 8                     if (!putnextctl(wrq, M_BREAK))
 9                                 num_alloc_fail++;
10                        break;
11
12                 case M_DELAY:
13                     if (!putnextctl1(wrq, M_DELAY, parm))
14                                 num_alloc_fail++;
15                     break;
```

EXAMPLE 1 *(Continued)*

```

            default:
                  cmn_err(CE_PANIC, "send_ctl: bad message type passed");
                     break;
               }
   }
```

SUMMARY OF TRUSTED SOLARIS CHANGES

The new message allocated by this routine is labeled with the default streams attributes of the target queue.

Trusted Solaris 8 4/01 Reference Manual

putctl(9F), putnextctl1(9F)

SunOS 5.8 Reference Manual

put(9E), cmn_err(9F), datamsg(9F)

Writing Device Drivers

STREAMS Programming Guide

NAME

putq – Put a message on a queue

SYNOPSIS

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

`int` **`putq`**`(queue_t *`*q*`, mblk_t *`*bp*`);`

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

PARAMETERS

q	Pointer to the queue to which the message is to be added.
bp	Message to be put on the queue.

DESCRIPTION

`putq()` is used to put messages on a driver's queue after the module's put routine has finished processing the message. The message is placed after any other messages of the same priority, and flow control parameters are updated. If `QNOENB` is not set, the service routine is enabled. If no other processing is done, `putq()` can be used as the module's put routine.

RETURN VALUES

`putq()` returns:

`1`	On success.
`0`	On failure.

CONTEXT

`putq()` can be called from user or interrupt context.

EXAMPLES

See the `datamsg`(9F) function page for an example of `putq()`.

SUMMARY OF TRUSTED SOLARIS CHANGES

This routine tries to assign attribute structures to the `mblks` of a message that does not have any. The first attribute stucture found is used. If a stream module illegally combined messages, `mblks` can have different attribute structures; in that case, the message might be dropped by this routine unless overridden by the `TSOL_STR_LINKB` flag.

Trusted Solaris 8 4/01 Reference Manual

`putbq`(9F)

SunOS 5.8 Reference Manual

`datamsg`(9F), `qenable`(9F), `rmvq`(9F)

Writing Device Drivers, *STREAMS Programming Guide*

NOTES

These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

NAME

tsol_get_strattr, tsol_set_strattr – Get security attributes of a message

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/tsol/tstream.h>

tsol_str_attr *tsol_get_strattr(mblk_t *mp);

void tsol_set_strattr(mblk_t *mp, tsol_strattr_t *strattr);
```

PARAMETERS

mp	Pointer to message block
strattr	Pointer to a STREAMS attributes structure

DESCRIPTION

`tsol_get_strattr()` is called by a STREAMS routine to find the attributes attached to a STREAMS message. `tsol_get_strattr()` returns a pointer to the first attibute structure found. The attribute-structure reference count will be increased to prevent the structure's being released. A module using this routine must call the `SATTR_RELE` macro to free the attribute structure. If no attribute structure is found in a message, `NULL` is returned.

`tsol_set_strattr()` replaces the current set of STREAMS attributes in the message to which *mp* points with new attributes supplied by *strattr*. If either *mp* or *strattr* is set to `NULL`, the function exits without changing *mp*.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Trusted Solaris only
Interface-Level	Architecture-independent

CONTEXT

This routine can be called from the interrupt level. The routine will not sleep. The reference count is protected by a short-term spin-lock mutex.

NOTES

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME

tsol_get_strattr, tsol_set_strattr – Get security attributes of a message

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/tsol/tstream.h>
```

`tsol_str_attr *`**`tsol_get_strattr`**`(mblk_t *`*mp*`);`

`void `**`tsol_set_strattr`**`(mblk_t *`*mp*`, tsol_strattr_t *`*strattr*`);`

PARAMETERS

mp	Pointer to message block
strattr	Pointer to a STREAMS attributes structure

DESCRIPTION

`tsol_get_strattr()` is called by a STREAMS routine to find the attributes attached to a STREAMS message. `tsol_get_strattr()` returns a pointer to the first attibute structure found. The attribute-structure reference count will be increased to prevent the structure's being released. A module using this routine must call the `SATTR_RELE` macro to free the attribute structure. If no attribute structure is found in a message, `NULL` is returned.

`tsol_set_strattr()` replaces the current set of STREAMS attributes in the message to which *mp* points with new attributes supplied by *strattr*. If either *mp* or *strattr* is set to `NULL`, the function exits without changing *mp*.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Trusted Solaris only
Interface-Level	Architecture-independent

CONTEXT

This routine can be called from the interrupt level. The routine will not sleep. The reference count is protected by a short-term spin-lock mutex.

NOTES

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

Index