



# man pages section 9F: Kernel Functions for Drivers

---

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

Part No: 817-0887-10  
May 2003

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPOUDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



030523 @ 5943



# Contents

---

**Preface**    5

**Introduction**    11

Intro(9F)    12

**Kernel Functions for Drivers**    31

copyb(9F)    32

copymsg(9F)    34

dupb(9F)    36

dupmsg(9F)    39

insq(9F)    40

kstat\_create(9F)    42

linkb(9F)    44

msgpullup(9F)    45

pullupmsg(9F)    46

put(9F)    48

putbq(9F)    49

putctl1(9F)    50

putctl(9F)    51

putnext(9F)    53

putnextctl1(9F)    54

putnextctl(9F)    55

putq(9F)    57

tsol\_get\_strattr(9F)    58

**Index    59**

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

#### OPERANDS

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

#### OUTPUT

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

#### RETURN VALUES

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

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

#### ERRORS

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

#### USAGE

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

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

#### EXAMPLES

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

---

## Intro(9F)

NAME	Intro – introduction to DDI/DKI functions
DESCRIPTION	<p>Section 9F describes the Solaris and Trusted Solaris kernel functions available for use by device drivers. See <code>Intro(9)</code> for how Trusted Solaris man pages are included in section 9F.</p> <p><b>Note</b> – The printed <i>Trusted Solaris 8 HW 12/02 Reference Manual</i> 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 <i>SunOS 5.8 Reference Manual</i>. For a fuller description, see Trusted Solaris Manual Page Display in <code>Intro(1)</code>. The SEE ALSO man page heading has been subdivided to help users of the printed manual locate a referenced man page.</p>
HEADINGS	<p>In this section, the information for each driver function is organized under the following headings:</p> <ul style="list-style-type: none"> <li>■ NAME summarizes the function’s purpose.</li> <li>■ SYNOPSIS shows the syntax of the function’s entry point in the source code. #include directives are shown for required headers.</li> <li>■ INTERFACE LEVEL describes any architecture dependencies.</li> <li>■ ARGUMENTS describes any arguments required to invoke the function.</li> <li>■ DESCRIPTION describes general information about the function.</li> <li>■ RETURN VALUES describes the return values and messages that can result from invoking the function.</li> <li>■ CONTEXT indicates from which driver context (user, kernel, interrupt, or high-level interrupt) the function can be called.</li> <li>■ A driver function has <i>user context</i> if it was directly invoked because of a user thread. The <code>read(9E)</code> entry point of the driver, invoked by a <code>read(2)</code> system call, has user context.</li> <li>■ A driver function has <i>kernel context</i> if was invoked by some other part of the kernel. In a block device driver, the <code>strategy(9E)</code> 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 <code>strategy(9E)</code> has kernel context.</li> <li>■ <i>Interrupt context</i> is kernel context, but also has an interrupt level associated with it. Driver interrupt routines have interrupt context.</li> <li>■ <i>High-level interrupt context</i> is a more restricted form of interrupt context. If <code>ddi_intr_hilevel(9F)</code> indicates that an interrupt is high-level, driver interrupt routines added for that interrupt with <code>ddi_add_intr(9F)</code> run in high-level interrupt context. These interrupt routines are only allowed to call <code>ddi_trigger_softintr(9F)</code>, <code>mutex_enter(9F)</code> and <code>mutex_exit(9F)</code>. Furthermore, <code>mutex_enter(9F)</code> and <code>mutex_exit(9F)</code> may only be called on mutexes initialized with the <code>ddi_iblock_cookie</code> returned by <code>ddi_get_iblock_cookie(9F)</code>.</li> <li>■ SEE ALSO indicates functions that are related by usage and sources, and which can be referred to for further information.</li> </ul>

**STREAMS Kernel  
Function Summary**

- 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.

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

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

Routine	Type
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
rmvq	DDI/DKI

<b>Routine</b>	<b>Type</b>
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.

<b>Routine</b>	<b>Type</b>
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
btop	DDI/DKI

<b>Routine</b>	<b>Type</b>
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
ddi_dma_alloc_handle	Solaris DDI



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

<b>Routine</b>	<b>Type</b>
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
ddi_intr_hilevel	Solaris DDI

<b>Routine</b>	<b>Type</b>
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
ddi_mapdev	Solaris DDI

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

<b>Routine</b>	<b>Type</b>
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
ddi_prop_exists	Solaris DDI

<b>Routine</b>	<b>Type</b>
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
ddi_regs_map_setup	Solaris DDI

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

Routine	Type
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
inb	Solaris x86 DDI



Routine	Type
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
mutex_enter	Solaris DDI

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

<b>Routine</b>	<b>Type</b>
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
scsi_dmafree	Solaris DDI

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

<b>Routine</b>	<b>Type</b>
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_try	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
va_start	Solaris DDI

## Intro(9F)

Routine	Type
<code>vcmn_err</code>	DDI/DKI
<code>vsprintf</code>	Solaris DDI

## Kernel Functions for Drivers

---

copyb(9F)

<b>NAME</b>	copyb – copy a message block
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  mblk_t *copyb(mblk_t *bp);</pre>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>PARAMETERS</b>	<i>bp</i> Pointer to the message block from which data is copied.
<b>DESCRIPTION</b>	<p>copyb() allocates a new message block, and copies into it the data from the block that <i>bp</i> denotes. The new block will be at least as large as the block being copied. copyb() uses the <i>b_rptr</i> and <i>b_wptr</i> members of <i>bp</i> to determine how many bytes to copy.</p>
<b>RETURN VALUES</b>	If successful, copyb() returns a pointer to the newly allocated message block containing the copied data. Otherwise, it returns a NULL pointer.
<b>CONTEXT</b>	copyb() can be called from user or interrupt context.
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Using copyb</p> <p>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.</p> <p>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.</p> <pre> 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      ... 10     int             h_address; 11     ... 12 }; 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-&gt;r_outq)) {      /* no room */</pre>



**EXAMPLE 1** Using copyb (Continued)

```

22             ret = ret->r_next;
23             continue;
24         }
25         bp = copyb(header);           /* copy header msg. block */
26         if (bp == NULL)
27             break;
28         mp = dupmsg(ret->r_mp);         /* duplicate data */
29         if (mp == NULL) {             /* if unsuccessful */
30             freeb(bp);               /* free the block */
31             break;
32         }
33         php = (struct protoheader *)bp->b_rptr;
34         php->h_address = ret->r_address; /* new header */
35         bp->bp_cont = mp;             /* link the message */
36         putnext(ret->r_outq, bp);      /* send downstream */
37         ret = ret->r_next;
38     }
39     /* reschedule */
40     (void) timeout(retransmit, (caddr_t)ret, RETRANS_TIME);
41 }

```

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
Reference Manual**

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

dupmsg(9F)

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.

## copymsg(9F)

<b>NAME</b>	copymsg – Copy a message
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  mblk_t *copymsg(mblk_t *mp);</pre>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>PARAMETERS</b>	<i>mp</i> Pointer to the message to be copied.
<b>DESCRIPTION</b>	copymsg() forms a new message by allocating new message blocks and copying the contents of the message referred to by <i>mp</i> , using the copyb(9F) function. It returns a pointer to the new message.
<b>RETURN VALUES</b>	If the copy is successful, copymsg() returns a pointer to the new message. Otherwise, it returns a NULL pointer.
<b>CONTEXT</b>	copymsg() can be called from user or interrupt context.
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Using copymsg</p> <p>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).</p> <pre> 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-&gt;b_datap-&gt;db_ref &gt; 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-&gt;b_cont) { 16         for (cp = tmp-&gt;b_rptr; cp &lt; tmp-&gt;b_wptr; cp++) { 17             if ((*cp &lt;= 'z') &amp;&amp; (*cp &gt;= 'a')) 18                 *cp -= 0x20; 19         } 20     } 21     return(cmp); 22 }</pre>

EXAMPLE 1 Using copymsg (Continued)

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

copyb(9F)  
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.

dupb(9F)

<b>NAME</b>	dupb – Duplicate a message block descriptor
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  mblk_t *dupb(mblk_t *bp);</pre>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>PARAMETERS</b>	<p><i>bp</i>            Pointer to the message block to be duplicated. mblk_t is an instance of the msgb(9S) structure.</p>
<b>DESCRIPTION</b>	<p>dupb() creates a new mblk_t structure (see msgb(9S)) to reference the message block pointed to by <i>bp</i>.</p> <p>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 <i>bp</i>.</p>
<b>RETURN VALUES</b>	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).
<b>CONTEXT</b>	dupb() can be called from user, kernel, or interrupt context.
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Using dupb()</p> <p>This <i>srv(9E)</i> (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.</p>

**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,
```

dupb(9F)

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

```
31             BPRI_MED, xxxcallback, (intptr_t)q);
32             return;
33         }
34         linkb(bp, mp);
35         putnext(q, bp);
36     } else {
37         putbq(q, mp);
38         return;
39     }
40 }
41 }
42 }
43 void
44 xxxcallback(q)
45     queue_t *q;
46 {
47     struct xx *xx = (struct xx *)q->q_ptr;
48
49     xx->xx_qbufcall_id = 0;
50     qenable(q);
51 }
52
53 xxxclose(q, cflag, crp)
54     queue_t *q;
55     int cflag;
56     cred_t *crp;
57 {
58     struct xx *xx = (struct xx *)q->q_ptr;
59     ...
60     if (xx->xx_qbufcall_id)
61         qunbufcall(q, xx->xx_qbufcall_id);
62     ...
63 }
```

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
Release Notes  
Reference Manual

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

copyb(9F)

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.

<b>NAME</b>	dupmsg – Duplicate a message
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  mblk_t *dupmsg(mblk_t *mp);</pre>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>PARAMETERS</b>	<i>mp</i> Pointer to the message.
<b>DESCRIPTION</b>	dupmsg() forms a new message by copying the message block descriptors pointed to by <i>mp</i> and linking them. dupb(9F) is called for each message block. The data blocks themselves are not duplicated.
<b>RETURN VALUES</b>	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).
<b>CONTEXT</b>	dupmsg() can be called from user, kernel, or interrupt context.
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Using dupmsg</p> <p>See copyb(9F) for an example using dupmsg().</p>
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	This routine will preserve the attributes of the message it manipulates.
<b>SEE ALSO</b>	copyb(9F), copymsg(9F), dupb(9F)
<b>REFERENCES</b>	datab(9S)
<b>Reference Manual</b>	<p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
<b>NOTES</b>	These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

## insq(9F)

<b>NAME</b>	insq – Insert a message into a queue
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  int <b>insq</b>(queue_t *q, mblk_t *emp, mblk_t *nmp);</pre>
<b>PARAMETERS</b>	<p><i>q</i>            Pointer to the queue containing message <i>emp</i>.</p> <p><i>emp</i>          Enqueued message before which the new message is to be inserted. mblk_t is an instance of the msgb(9S) structure.</p> <p><i>nmp</i>          Message to be inserted.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	insq() inserts a message into a queue. The message to be inserted, <i>nmp</i> , is placed in <i>q</i> immediately before the message <i>emp</i> . If <i>emp</i> 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.
<b>RETURN VALUES</b>	insq() returns: <p>1            On success.</p> <p>0            On failure.</p>
<b>CONTEXT</b>	insq() can be called from user or interrupt context.
<b>EXAMPLES</b>	<p>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, <i>bp</i> will be NULL and we fall out of the for loop (line 21). insq() acts like putq(9F) in this case.</p> <pre> 1  #include &lt;sys/tihdr.h&gt; 2  #include &lt;sys/stream.h&gt; 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-&gt;b_datap-&gt;db_type) { 12      case M_PROTO: 13          tp = (union T_primitives *)mp-&gt;b_rptr; 14          switch (tp-&gt;type) { 15          case T_DATA_REQ: 16              putq(q, mp); 17              break; 18 19          case T_EXDATA_REQ:</pre>



insq(9F)

```

20         freezestr(q);
21         for (bp = q->q_first; bp; bp = bp->b_next) {
22             if (bp->b_datap->db_type == M_PROTO) {
23                 ntp = (union T_primitives *)bp->b_rptr;
24                 if (ntp->type != T_EXDATA_REQ)
25                     break;
26             }
27         }
28         (void)insq(q, bp, mp);
29         unfreezestr(q);
30         break;
31     . . . }
32 }
33 }
```

# 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  
HW 12/02  
Reference Manual

Intro(9F), putq(9F)  
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

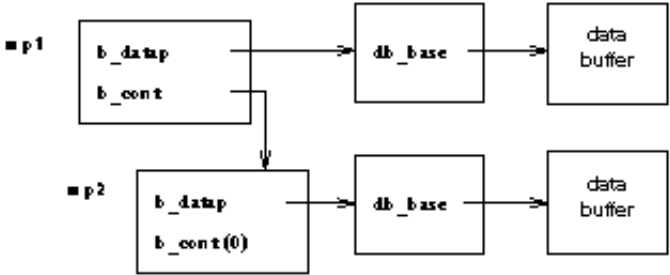
The stream must be frozen using freezestr(9F) before calling insq().  
These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

## kstat\_create(9F)

<b>NAME</b>	kstat_create – Create and initialize a new kstat														
<b>SYNOPSIS</b>	<pre>#include &lt;sys/types.h&gt; #include &lt;sys/kstat.h&gt;  kstat_t *kstat_create(char *module, int instance, char *name, char     *class, uchar_t type, ulong_t ndata, uchar_t ks_flag);</pre>														
<b>INTERFACE LEVEL PARAMETERS</b>	<p>Solaris DDI specific (Solaris DDI)</p> <table> <tr> <td><i>module</i></td><td>The name of the provider's module (such as "sd", "esp", ...). The "core" kernel uses the name "unix".</td></tr> <tr> <td><i>instance</i></td><td>The provider's instance number, as from ddi_get_instance(9F). Modules which do not have a meaningful instance number should use 0.</td></tr> <tr> <td><i>name</i></td><td>A pointer to a string that uniquely identifies this structure. Only KSTAT_STRLEN – 1 characters are significant.</td></tr> <tr> <td><i>class</i></td><td>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.</td></tr> <tr> <td><i>type</i></td><td> <p>The type of kstat to allocate. Valid types are:</p> <p>KSTAT_TYPE_NAMED Allows more than one data record per kstat.</p> <p>KSTAT_TYPE_INTR Interrupt; only one data record per kstat.</p> <p>KSTAT_TYPE_IO I/O; only one data record per kstat</p> </td></tr> <tr> <td><i>ndata</i></td><td>The number of type-specific data records to allocate.</td></tr> <tr> <td><i>flag</i></td><td> <p>A bit-field of various flags for this kstat. <i>flag</i> is some combination of:</p> <p>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.</p> <p>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.</p> <p>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</p> </td></tr> </table>	<i>module</i>	The name of the provider's module (such as "sd", "esp", ...). The "core" kernel uses the name "unix".	<i>instance</i>	The provider's instance number, as from ddi_get_instance(9F). Modules which do not have a meaningful instance number should use 0.	<i>name</i>	A pointer to a string that uniquely identifies this structure. Only KSTAT_STRLEN – 1 characters are significant.	<i>class</i>	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.	<i>type</i>	<p>The type of kstat to allocate. Valid types are:</p> <p>KSTAT_TYPE_NAMED Allows more than one data record per kstat.</p> <p>KSTAT_TYPE_INTR Interrupt; only one data record per kstat.</p> <p>KSTAT_TYPE_IO I/O; only one data record per kstat</p>	<i>ndata</i>	The number of type-specific data records to allocate.	<i>flag</i>	<p>A bit-field of various flags for this kstat. <i>flag</i> is some combination of:</p> <p>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.</p> <p>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.</p> <p>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</p>
<i>module</i>	The name of the provider's module (such as "sd", "esp", ...). The "core" kernel uses the name "unix".														
<i>instance</i>	The provider's instance number, as from ddi_get_instance(9F). Modules which do not have a meaningful instance number should use 0.														
<i>name</i>	A pointer to a string that uniquely identifies this structure. Only KSTAT_STRLEN – 1 characters are significant.														
<i>class</i>	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.														
<i>type</i>	<p>The type of kstat to allocate. Valid types are:</p> <p>KSTAT_TYPE_NAMED Allows more than one data record per kstat.</p> <p>KSTAT_TYPE_INTR Interrupt; only one data record per kstat.</p> <p>KSTAT_TYPE_IO I/O; only one data record per kstat</p>														
<i>ndata</i>	The number of type-specific data records to allocate.														
<i>flag</i>	<p>A bit-field of various flags for this kstat. <i>flag</i> is some combination of:</p> <p>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.</p> <p>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.</p> <p>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</p>														

	kstat_create(9F)
	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	<p>kstat_create() is used in conjunction with kstat_install(9F) to allocate and initialize a kstat(9S) structure. The method is generally as follows:</p> <p>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.</p> <p>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.</p> <p>Once the kstat is completely initialized, kstat_install(9F) is used to make the kstat accessible to the outside world.</p>
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	<p><b>EXAMPLE 1</b> Allocating and Initializing a kstat Structure</p> <pre> pkstat_t      *ksp; ksp = kstat_create(module, instance, name, class, type, ndata, flags); if (ksp) {     /* ... provider initialization, if necessary */     kstat_install(ksp); } </pre>
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 HW 12/02 Reference Manual	<p>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.</p> <p>Intro(9F)</p> <p>kstat(3KSTAT), ddi_get_instance(9F), attributes(5), kstat_delete(9F), kstat_install(9F), kstat_named_init(9F), kstat(9S), kstat_named(9S)</p> <p><i>Writing Device Drivers</i></p>
NOTES	These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

linkb(9F)

NAME	linkb – concatenate two message blocks
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt;  void linkb(mblk_t *mp1, mblk_t *mp2);</pre>
PARAMETERS	<p><i>mp1</i>        The message to which <i>mp2</i> is to be added. mblk_t is an instance of the msgb(9S) structure.</p> <p><i>mp2</i>        The message to be added.</p>
INTERFACE LEVEL	Architecture independent level 1 (DDI/DKI).
DESCRIPTION	<p>linkb() creates a new message by adding <i>mp2</i> to the tail of <i>mp1</i>. The continuation pointer, <i>b_cont</i>, of <i>mp1</i> is set to point to <i>mp2</i>.</p>  <pre> graph LR     mp1["mp1 b_datap b_cont"] --&gt; db_base1["db_base"]     db_base1 --&gt; data_buffer1["data buffer"]     mp2["mp2 b_datap b_cont(0)"] --&gt; db_base2["db_base"]     db_base2 --&gt; data_buffer2["data buffer"]     mp1 -- b_cont --&gt; mp2 -- b_datap </pre> <p>linkb(mp1, mp2);</p>
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 <i>mp1</i> and <i>mp2</i> are equivalent before calling this function. Mixing messages with non-equivalent attributes can result in a violation of the security policy.
Trusted Solaris 8 HW 12/02 Reference Manual	<p>dupb(9F), unlinkb(9F)</p> <p>msgb(9S)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>

<b>NAME</b>	msgpullup – Concatenate bytes in a message
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  mblk_t *msgpullup(mblk_t *mp, ssize_t len);</pre>
<b>PARAMETERS</b>	<p><i>mp</i>            Pointer to the message whose blocks are to be concatenated.</p> <p><i>len</i>            Number of bytes to concatenate.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	msgpullup() concatenates and aligns the first <i>len</i> data bytes of the message pointed to by <i>mp</i> , 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 <i>len</i> equals -1, all data are concatenated. If <i>len</i> bytes of the same message type cannot be found, msgpullup() fails and returns NULL.
<b>RETURN VALUES</b>	Upon successful completion, msgpullup() returns a pointer to the new message. Upon failure, msgpullup() returns NULL.
<b>CONTEXT</b>	msgpullup() can be called from user or interrupt context.
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	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.
<b>Trusted Solaris 8 HW 12/02 Reference Manual</b>	<p>intro(9F), pullupmsg(9F)</p> <p>srv(9E), allocb(9F), msgb(9S)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
<b>NOTES</b>	<p>msgpullup() is a DKI-compliant replacement for the older pullupmsg(9F) routine. Users are strongly encouraged to use msgpullup() instead of pullupmsg(9F).</p> <p>These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.</p>

## pullupmsg(9F)

<b>NAME</b>	pullupmsg – Concatenate bytes in a message
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  int <b>pullupmsg</b>(mblk_t *mp, ssize_t len);</pre>
<b>PARAMETERS</b>	<p><i>mp</i>            Pointer to the message whose blocks are to be concatenated. mblk_t is an instance of the msgb(9S) structure.</p> <p><i>len</i>            Number of bytes to concatenate.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	<p>pullupmsg() tries to combine multiple data blocks into a single block. pullupmsg() concatenates and aligns the first <i>len</i> data bytes of the message pointed to by <i>mp</i>. If <i>len</i> equals -1, all data are concatenated. If <i>len</i> bytes of the same message type cannot be found, pullupmsg() fails and returns 0.</p>
<b>RETURN VALUES</b>	<p>pullupmsg() returns:</p> <p>1            On success.</p> <p>0            On failure.</p>
<b>CONTEXT</b>	pullupmsg() can be called from user or interrupt context.
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Using pullupmsg</p> <p>This is a driver write <i>srv</i>(9E) (service) routine for a device that does not support scatter/gather DMA. For all <i>M_DATA</i> messages, the data will be transferred to the device with DMA.</p> <p>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.</p> <pre> 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-&gt;b_datap-&gt;db_type) { 11             case M_DATA: 12                 if (pullupmsg(mp, -1)) { 13                     dma_addr = vtop(mp-&gt;b_rptr); 14                     dma_len = mp-&gt;b_wptr - mp-&gt;b_rptr; 15                     xxx_do_dma(dma_addr, dma_len); 16                     freemsg(mp); </pre>

**EXAMPLE 1** Using pullupmsg (Continued)

```
17             break;
18         }
19         for (tmp = mp; tmp; tmp = tmp->b_cont) {
20             dma_addr = vtop(tmp->b_rptr);
21             dma_len = tmp->b_wptr - tmp->b_rptr;
22             xxx_do_dma(dma_addr, dma_len);
23         }
24         freemsg(mp);
25         break;
26     . . .
27 }
28 }
```

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

**Trusted Solaris 8  
HW 12/02  
Reference Manual**

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.

Intro(9F), msgpullup(9F)  
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.

put(9F)

<b>NAME</b>	put – Call a STREAMS put () procedure
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt; #include &lt;sys/ddi.h&gt;  void <b>put</b> (queue_t *q, mblk_t *mp);</pre>
<b>PARAMETERS</b>	<p><i>q</i>            Pointer to a STREAMS queue.</p> <p><i>mp</i>           Pointer to message block being passed into queue.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	put () calls the put procedure (put(9E) entry point) for the STREAMS queue specified by <i>q</i> , passing it the message block referred to by <i>mp</i> . It is typically used by a driver or module to call its own put () procedure.
<b>CONTEXT</b>	<p>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 <i>q</i> argument.</p> <p>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.</p>
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	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.
<b>Trusted Solaris 8 HW 12/02 Reference Manual</b>	<p>putctl(9F), putctl1(9F), putnext(9F), putnextctl(9F), putnextctl1(9F)</p> <p>put(9E), freezestr(9F), qreply(9F)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
<b>NOTES</b>	<p>The caller cannot have the stream frozen when calling this function. See freezestr(9F).</p> <p>These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.</p> <p>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.</p>



NAME	putbq – place a message at the head of a queue
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt;  int putbq(queue_t *q, mblk_t *bp);</pre>
INTERFACE LEVEL	Architecture independent level 1 (DDI/DKI).
PARAMETERS	<p><i>q</i>            Pointer to the queue.</p> <p><i>bp</i>           Pointer to the message block.</p>
DESCRIPTION	<p>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.</p> <p>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.</p>
RETURN VALUES	<p>putbq() returns 1 upon success and 0 upon failure.</p> <p><b>Note</b> – Upon failure, the caller should call freemsg(9F) to free the pointer to the message block.</p>
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 HW 12/02 Reference Manual	<p>putq(9F)</p> <p>bcanput(9F), bufcall(9F), canput(9F), freemsg(9F), getq(9F)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
NOTES	These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

## putctl1(9F)

<b>NAME</b>	putctl1 – send a control message with a one-byte parameter to a queue
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  int <b>putctl1</b>(queue_t *q, int type, int p);</pre>
<b>PARAMETERS</b>	<p><i>q</i> Queue to which the message is to be sent.</p> <p><i>type</i> Type of message.</p> <p><i>p</i> One-byte parameter.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	<p>putctl1(), like putctl(9F), tests the <i>type</i> argument to make sure a data type has not been specified, and attempts to allocate a message block. The <i>p</i> 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 <i>type</i> 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 <i>q</i> with the newly allocated and initialized message.</p>
<b>RETURN VALUES</b>	On success, 1 is returned. 0 is returned if <i>type</i> is a data type or if a message block cannot be allocated.
<b>CONTEXT</b>	putctl1() can be called from user or interrupt context.
<b>EXAMPLES</b>	See the putctl(9F) function page for an example of putctl1().
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	The new message allocated by this routine is labeled with the default streams attributes of the target queue.
<b>Trusted Solaris 8 HW 12/02 Reference Manual</b>	<p>putctl(9F), putnextctl1(9F)</p> <p>put(9E), allocb(9F), datamsg(9F)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>

NAME	putctl – send a control message to a queue
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt;  int putctl(queue_t *q, int type);</pre>
PARAMETERS	<p><i>q</i> Queue to which the message is to be sent.</p> <p><i>type</i> Message type (must be control, not data type).</p>
INTERFACE LEVEL	Architecture independent level 1 (DDI/DKI).
DESCRIPTION	putctl() tests the <i>type</i> argument to make sure a data type has not been specified, and then attempts to allocate a message block. putctl() fails if <i>type</i> 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 <i>q</i> with the newly allocated and initialized messages.
RETURN VALUES	On success, 1 is returned. If <i>type</i> 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	<p><b>EXAMPLE 1</b> Using putctl</p> <p>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 <i>parm</i> 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).</p> <pre> 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-&gt;q_next, M_BREAK)) 12                num_alloc_fail++; 13            break; 14 15        case M_DELAY: 16            if (!putctl1(wrq-&gt;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;</pre>

putctl(9F)

**EXAMPLE 1** Using putctl (Continued)

```
23         }  
24     }
```

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
HW 12/02  
Reference Manual  
SunOS 5.8  
Reference Manual

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

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

cmn\_err(9F), datamsg(9F), put(9E)

*Writing Device Drivers*

*STREAMS Programming Guide*

<b>NAME</b>	putnext – Send a message to the next queue
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt; #include &lt;sys/ddi.h&gt;  void <b>putnext</b> (queue_t *q, mblk_t *mp);</pre>
<b>PARAMETERS</b>	<p><i>q</i>            Pointer to the queue from which the message <i>mp</i> will be sent.</p> <p><i>mp</i>           Message to be passed.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	putnext () is used to pass a message to the put(9E) routine of the next queue in the stream.
<b>RETURN VALUES</b>	None.
<b>CONTEXT</b>	putnext () can be called from user or interrupt context.
<b>EXAMPLES</b>	See allocb(9F) for an example of using putnext () .
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	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.
<b>Trusted Solaris 8 HW 12/02 Reference Manual</b>	<p>Intro(9F)</p> <p>allocb(9F), put(9E)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
<b>NOTES</b>	These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

## putnextctl1(9F)

<b>NAME</b>	putnextctl1 – send a control message with a one-byte parameter to a queue
<b>SYNOPSIS</b>	<pre>#include &lt;sys/stream.h&gt;  int <b>putnextctl1</b>(queue_t *q, int type, int p);</pre>
<b>PARAMETERS</b>	<p><i>q</i> Queue to which the message is to be sent.</p> <p><i>type</i> Type of message.</p> <p><i>p</i> One-byte parameter.</p>
<b>INTERFACE LEVEL</b>	Architecture independent level 1 (DDI/DKI).
<b>DESCRIPTION</b>	<p><code>putnextctl1()</code>, like <code>putctl1(9F)</code>, tests the <i>type</i> argument to make sure a data type has not been specified, and attempts to allocate a message block. The <i>p</i> parameter can be used, for example, to specify how long the delay will be when an <code>M_DELAY</code> message is being sent. <code>putnextctl1()</code> fails if <i>type</i> is <code>M_DATA</code>, <code>M_PROTO</code>, or <code>M_PCPROTO</code>, or if a message block cannot be allocated. If successful, <code>putnextctl1()</code> calls the <code>put(9E)</code> routine of the queue pointed to by <i>q</i> with the newly allocated and initialized message.</p> <p>A call to <code>putnextctl1(q, type, p)</code> is an atomic equivalent of <code>putctl1(q-&gt;q_next, type, p)</code>. The STREAMS framework provides whatever mutual exclusion is necessary to insure that de-referencing <i>q</i> through its <code>q_next</code> field and then invoking <code>putctl1(9F)</code> proceeds without interference from other threads.</p> <p><code>putnextctl1(9F)</code> <code>putctl1(9F)</code>.</p>
<b>RETURN VALUES</b>	On success, 1 is returned. 0 is returned if <i>type</i> is a data type, or if a message block cannot be allocated.
<b>CONTEXT</b>	<code>putnextctl1()</code> can be called from user or interrupt context.
<b>EXAMPLES</b>	See the <code>putnextctl1(9F)</code> function page for an example of <code>putnextctl1()</code> .
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	The new message allocated by this routine is labeled with the default streams attributes of the target queue.
<b>Trusted Solaris 8 HW 12/02 Reference Manual</b>	<p><code>putctl1(9F)</code>, <code>putctl1(9F)</code>, <code>putnextctl1(9F)</code></p> <p><code>alloca(9F)</code>, <code>put(9E)</code>, <code>datamsg(9F)</code></p> <p><i>Writing Device Drivers, STREAMS Programming Guide</i></p>

NAME	putnextctl – send a control message to a queue
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt;  int putnextctl(queue_t *q, int type);</pre>
PARAMETERS	<p><i>q</i> Queue to which the message is to be sent.</p> <p><i>type</i> Message type (must be control, not data type).</p>
INTERFACE LEVEL	Architecture independent level 1 (DDI/DKI).
DESCRIPTION	<p>putnextctl() tests the <i>type</i> argument to make sure a data type has not been specified, and then attempts to allocate a message block. putnextctl() fails if <i>type</i> 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 <i>q</i> with the newly allocated and initialized messages.</p> <p>A call to putnextctl(<i>q</i>, <i>type</i>) is an atomic equivalent of putctl(<i>q</i>→<i>q_next</i>, <i>type</i>). The STREAMS framework provides whatever mutual exclusion is necessary to ensure that dereferencing <i>q</i> through its <i>q_next</i> field and then invoking putctl(9F) proceeds without interference from other threads.</p> <p>putnextctl() should always be used in preference to putctl(9F).</p>
RETURN VALUES	On success, 1 is returned. If <i>type</i> 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	<p><b>EXAMPLE 1</b> Using putnextctl() in a send_ctl() Routine</p> <p>The send_ctl() routine is used to pass control messages downstream. messages are handled with putnextctl() (line 8). putnextctl(9F) (line 13) is used for M_DELAY messages, so that <i>parm</i> 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).</p> <pre> 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;</pre>

putnextctl(9F)

**EXAMPLE 1** Using putnextctl() in a send\_ctl() Routine (Continued)

```
16
17         default:
18             cmn_err(CE_PANIC, "send_ctl: bad message type passed");
19             break;
20         }
21     }
```

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
HW 12/02  
Reference Manual  
SunOS 5.8  
Reference Manual

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

putctl(9F), putnextctl(9F)

put(9E), cmn\_err(9F), datamsg(9F)

*Writing Device Drivers*

*STREAMS Programming Guide*



NAME	putq – put a message on a queue
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt;  int <b>putq</b>(queue_t *q, mblk_t *bp);</pre>
INTERFACE LEVEL	Architecture independent level 1 (DDI/DKI).
PARAMETERS	<p><i>q</i>            Pointer to the queue to which the message is to be added.</p> <p><i>bp</i>           Message to be put on the queue.</p>
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	<p>putq() returns 1 on success and 0 on failure.</p> <p><b>Note</b> – Upon failure, the caller should call freemsg(9F) to free the pointer to the message block.</p>
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 structure 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 HW 12/02 Reference Manual	<p>putbq(9F)</p> <p>datamsg(9F), freemsg(9F), qenable(9F), rmvq(9F)</p> <p><i>Writing Device Drivers</i></p> <p><i>STREAMS Programming Guide</i></p>
NOTES	These interfaces are uncommitted. Although not expected to do so, they may change between minor Trusted Solaris releases.

## tsol\_get\_strattr(9F)

NAME	tsol_get_strattr, tsol_set_strattr – Get security attributes of a message						
SYNOPSIS	<pre>#include &lt;sys/stream.h&gt; #include &lt;sys/tsol/tstream.h&gt;  tsol_str_attr *tsol_get_strattr(mblk_t *mp); void tsol_set_strattr(mblk_t *mp, tsol_strattr_t *strattr);</pre>						
PARAMETERS	<p><i>mp</i>                      Pointer to message block</p> <p><i>strattr</i>                Pointer to a STREAMS attributes structure</p>						
DESCRIPTION	<p>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 attribute 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.</p> <p>tsol_set_strattr() replaces the current set of STREAMS attributes in the message to which <i>mp</i> points with new attributes supplied by <i>strattr</i>. If either <i>mp</i> or <i>strattr</i> is set to NULL, the function exits without changing <i>mp</i>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>Trusted Solaris only</td></tr><tr><td>Interface-Level</td><td>Architecture-independent</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	Trusted Solaris only	Interface-Level	Architecture-independent
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

---

## C

call a STREAMS put procedure — `put`, 48  
`copyb` — copy a message block, 32  
`copymsg` — copy a message, 34  
create and initialize a new `kstat` —  
    `kstat_create`, 43

## D

`dupb` — duplicate a message block  
    descriptor, 36  
duplicate a message — `dupmsg`, 39  
duplicate a message block descriptor —  
    `dupb`, 36  
`dupmsg` — duplicate a message, 39

## I

`insq` — insert a message into a queue, 40

## K

`kstat_create` — create and initialize a new  
    `kstat`, 43

## L

`linkb` — concatenate two message blocks, 44

## M

`msgpullup` — concatenate bytes in a  
    message, 45

## P

`pullupmsg` — concatenate bytes in a  
    message, 46  
`put` — call a STREAMS put procedure, 48  
`putbq` — place a message at the head of a  
    queue, 49  
`putctl` — send a control message to a queue, 51  
`putctl1` — send a control message with a  
    one-byte parameter to a queue, 50  
`putnext` — send a message to the next  
    queue, 53  
`putnextctl` — send a control message to a  
    queue, 55  
`putnextctl1` — send a control message with a  
    one-byte parameter to a queue, 54  
`putq` — put a message on a queue, 57

## S

STREAMS message blocks  
    concatenate bytes in a message —  
        `msgpullup`, 45  
    concatenate bytes in a message —  
        `pullupmsg`, 46  
    concatenate two — `linkb`, 44  
    copy — `copyb`, 32

STREAMS message queue, insert a message into  
a queue — `insq`, 40

STREAMS messages, copy a message —  
`copymsg`, 34

STREAMS queues

- place a message at the head of a queue —  
`putbq`, 49
- put a message on a queue — `putq`, 57
- send a control message to a queue —  
`putctl`, 51
- send a control message to a queue —  
`putnextctl`, 55
- send a control message with a one-byte  
parameter to a queue — `putctl1`, 50
- send a control message with a one-byte  
parameter to a queue — `putnextctl1`, 54
- send a message to the next queue —  
`putnext`, 53

## T

`tsol_get_strattr` — get security attributes of a  
message, 58

`tsol_set_strattr` — set security attributes of a  
message, 58