

Trusted Solaris 2.5.1 Man Pages: 3CTSOL, 3KTSOL, 3NTSOL, 3RTSOL, 3TSOL, and 3X11TSOL Library Functions

Sun Microsystems Federal, Inc.
A Sun Microsystems, Inc. Business
901 San Antonio Road, MS USJC01-201
Palo Alto, CA 94303
U.S.A.

Copyright 1998 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 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.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

Sun, Sun Microsystems, the Sun logo, SunSoft, Solaris, SunOS, OpenWindows, DeskSet, ONC, ONC+, and NFS are trademarks, or registered trademarks 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.

RESTRICTED RIGHTS : Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

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 1997 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 Etats-Unis. 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 des systèmes 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, SunSoft, Solaris, SunOS, OpenWindows, DeskSet, ONC, ONC+, et NFS sont des marques de fabrique ou des marques déposées, 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 REPONDRE 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.

Portions © AT&T 1983-1990 and reproduced with permission from AT&T.

Preface

The Trusted Solaris operating environment is based on the SunOS operating system and other components of the Solaris operating environment and also bundles security-enhanced versions of the Common Desktop Environment (CDE), X windows, and Solstice AdminSuite tools. Therefore, the Trusted Solaris Reference Manual includes man pages not only for the operating system but also for the other products included in the Trusted Solaris product as well. In the Trusted Solaris Reference Manual, as in other UNIX reference manuals, each collection of information on a particular topic is called a man page, even though a man *page* may actually consist of *many pages* of text.

A man page is intended to answer concisely the question “What does it do”? Man pages are not intended to be tutorials. Depending what you are trying to do, refer to the Trusted Solaris user, developer, or administrator manuals for when and why to use a command or other features described in the man pages.

ACCESSING MAN PAGES

The man pages that make up the reference manual may be accessed in three ways.

Note: The following discussion of man page viewing options uses the term **package**, which is a unit of software typically delivered on a CD. Whoever installs a system usually decides whether or not all the packages are also installed. Installing the documentation packages is optional, because they are not required for operations. As a result, not everyone has access to every package. The packages that contain man pages in the Trusted Solaris operating environment are: SUNWman, plus SUNWaudmo , SUNWdtma , SUNWdtmad ,

SUNWkcsrt , SUNWkcspg , SUNWmfman , SUNWmfrun , SUNWolman , SUNWrtvcu , SUNWsadmm , SUNWtltkm , SUNWxwacx , SUNWxwman , SUNWxwplt , and SUNWxwpmn.

The first means of accessing the man pages is by using the **man**(1) command to view the man pages online. An account can use the **man** command when the man page package that contains the desired man page is available on the local system or mounted from a remote server, if, in addition, a terminal emulator (such as **dtterm**(1)) and the **man**(1) command are in one of the account's execution profiles. (For more about Trusted Solaris execution profiles and user accounts, see the Trusted Solaris user and administrator documentation.) To view a man page, enter the **man** command followed by the name of the man page. For example, to view the **ls**(1) man page that describes the command used to list directory's contents, a user enters the command: .

The second way to read man pages is by looking them up in the printed Trusted Solaris Reference Manual, which is in the Trusted Solaris documentation set, part number: TS2DS-251-9999.

The third means of reading the man pages is by viewing them in AnswerBook format. When the Trusted Solaris AnswerBook package, SUNWtab, is available on the local system or mounted from a remote server, anyone with the **answerbook**(1) command and a terminal emulator in an execution profile can display any of the man pages in the Trusted Solaris reference manual. The Trusted Solaris AnswerBook CD is packaged with the Trusted Solaris software CD. After the AnswerBook tool is launched, clicking the AnswerBook Navigator Search button brings up a dialog box where the name of a man page or terms contained in a man page can be entered to locate a specific man page.

For access to all available man pages for the operating system and for the bundled CDE, X windows, and Solstice AdminSuite products, the following man directories should be set in the MANPATH environment variable: **/usr/man**, **/usr/openwin/man**, and **/usr/dt/man**. For more about the format and contents of the man pages, see also the information in the **Intro** man pages for each section.

Trusted Solaris man pages are identified with a TSOL suffix in the section name. The TSOL suffix is used for man pages that are either added or modified from the base Solaris or bundled products.

- Section 1BTSOL describes printer commands adapted for Trusted Solaris from the Berkeley Software Distribution (BSD) print subsystem, which are used chiefly for printing administration.

Note: Use of the equivalent System V print commands is recommended (such as **lp**(1TSOL) instead of **lpr**(1BTSOL)) because although the BSD

commands are included for compatibility, they will be removed in future releases. Also, the BSD print management commands are not useful for managing print jobs on remote printers.

- Section 1MTSOL describes Trusted Solaris system maintenance and administration commands.
- Section 1TSOL describes modified user commands from the base SunOS operating system, and new Trusted Solaris user commands.
- Section 2TSOL describes Trusted Solaris new or modified system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- 3*TSOL subsections describe functions found in various Trusted Solaris libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2TSOL.

Subsections include: 3CTSOL, 3NTSOL, 3RTSOL, 3TSOL, and 3X11TSOL.

- Section 4TSOL outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5TSOL contains documentation for Trusted Solaris macros.
- 7*TSOL subsections describe various special files that refer to specific hardware peripherals and device drivers.

Subsections include: 7DTSOL, 7MTSOL, and 7TSOL.

- 9*TSOL subsections provide reference information for writing device drivers in the kernel operating system environment.

Trusted Solaris subsections are: 9FTSOL and 9TSOL.

Following is a generic list of headings on each man page. The man pages of each manual section include only the headings they need. 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. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. 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 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 3RTSOL to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

AVAILABILITY

This section briefly states any limitations on the availability of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1TSOL and Section 1MTSOL, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, the security administrator can use **pkgadd(1)** or **swmtool(1)** to install the missing package.

MT-LEVEL

This section lists the **MT-LEVEL** of the library functions described in the Section 3 manual pages. The **MT-LEVEL** defines the libraries' ability to support threads. See **Intro**(3TSOL) for more information.

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 7TSOL 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**(7).

OPTIONS

This 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 as **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 is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

- Commands**
- Modifiers**
- Variables**
- Expressions**
- Input Grammar**

EXAMPLES

This section provides examples of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. When an example is given for a command entered by a normal user, the prompt is shown as

example%

If the user must be in an administrative role, the example uses either the profile shell prompt for the secadmin or admin roles:

\$

or the root role prompt: **#**

Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT

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 filenames 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.

SEE ALSO

This section lists references to other man pages, in-house documentation, and outside publications.

DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

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.

SUMMARY OF TRUSTED SOLARIS CHANGES

On base man pages that have Trusted Solaris modifications, this section summarizes the changes described throughout the man page in a single easy-to-find place.

NAME	Intro, intro – introduction to functions and libraries
DESCRIPTION	This section describes functions found in various libraries, other than those functions that directly invoke basic SunOS or Trusted Solaris system primitives, which are described in Section 2 of the base reference manual and Section 2TSOL of the <i>Trusted Solaris Reference Manual</i>. Section 3 describes, in alphabetical order, all the functions available with the Trusted Solaris operating system, which is based on the SunOS operating system. Man pages whose section IDs end with the TSOL suffix (3CTSOL, 3KTSOL, 3NTSOL, 3RTSOL, 3TSOL, or 3X11TSOL) describe functions that are either new or modified from the base to work within Trusted Solaris security policy, including X library extensions. An example of a new Trusted Solaris function is labelinfo, which is described on the labelinfo(3TSOL) man page. The labelinfo system call is used to get information about security <i>labels</i> from the label_encodings(4TSOL) file. Modified functions are functions from any of the base products that have been modified to work within the Trusted Solaris <i>security policy</i>, such as: accept(3NTSOL)C . Man pages for modified functions have been rewritten to remove information that is not accurate for how the function behaves within the Trusted Solaris system. Modified man pages, such as accept, also have added descriptions for any new features and arguments added to the base. Function declarations can be obtained from the #include files indicated on each page. Certain major collections are identified by a letter after the section number: (3CTSOL) These functions in libc have been modified for Trusted Solaris. Some functions require privileges or otherwise behave differently from those in the SunOS libc libraries. Changes in behavior or requirements are noted on the individual manual pages. (3KTSOL) These Trusted Solaris new or modified functions allow access to the kernel's virtual memory library, which is implemented as a shared object, libkvm.so, but is not automatically linked by the C compilation system. Specify -lkvm on the cc command line to link with this library. (3NTSOL) The functions in this section are either in the Network Service Library, libnsl, or in the Trusted Systems Interoperability Group (TSIG) TSIX [RE] 1.1 library, libt6. libnsl.so and libt6.so are implemented as shared objects, and libnsl.a is also specified as an archive. Neither library is automatically linked by the C compilation system. Specify -lnsl on the cc command line to link with libnsl. Specify -lt6 on the cc command line to link with the libt6.so library. Some of the functions documented in 3NTSOL may incorporate other network libraries, including: • libsocket, • libresolv, • librpcsrv, • libnisdb,

- **librac**,
- **libxfn**, and
- **libkrb**.

(3RTSOL)

The functions described in this section are modified from the POSIX.4 Realtime library, **libposix4**. It is implemented only as a shared object, **libposix4.so**, and is not automatically linked by the C compilation system. Specify **-lposix4** on the **cc** command line to link with this library.

(3TSOL)

These functions constitute the Trusted Solaris library. **libtsol.so** is implemented as a shared object but is not automatically linked by the C compilation system. To link with the **libtsol** library specify **-ltsol** on the **cc** command line.

(3X11TSOL)

These functions constitute the Trusted Solaris extension to the X windows library. **libXtsol.so** is implemented as a shared object but is not automatically linked by the C compilation system. To link with the **libXtsol** library, specify **-lXtsol** after **-lX11** on the **cc** command line (**-lX11 -lXtsol**).

NOTE

System calls enforce policy for library routines, and you should generally look to the man page for the system call to find out how policy is enforced for the system call. However, policy is sometimes explained on the library routine man pages, according to the following guidelines:

- If the relationship between the library routine, and if the underlying system call is intuitively obvious [as is the relationship between **BR fopen** (3tsol) and **open**(2TSOL)Z, the related system call is mentioned in the **SEE ALSO** section, and the policy is not repeated on the library routine's man page.
- If the relationship between the library routine and the underlying system call(s) is not obvious [as is the relationship between **t6sendto**(3NTSOL) and **putpmsgattr**(2TSOL)], the policy information appears on the library routine's man page.
- If the relationship between the library routine and the underlying system call is clear, but the system call man page has so much information that the developer may have trouble finding it, the relevant information is repeated on the library routine's man page. An example is **t6peek_attr**(3NTSOL), which relies on **streamio**(7I), whose man page is 21 pages long.
- If the library is the exposed interface, and if the system call is undocumented, the policy appears on the library man page. An example is how some of the TSIX library routines rely on undocumented system calls.

DEFINITIONS

NOTE

See the **DEFINITIONS** section in the **Intro**(2TSOL) man page for terms and *Trusted Solaris Developer's Guide* for more information related to Trusted Solaris programming.

MT-Level of Libraries

A character is any bit pattern able to fit into a byte on the machine.

Exception: in some international languages, a “character” may require more than one byte, and is represented in multi-bytes.

The null character is a character with value 0, conventionally represented in the C language as `\0`. A character array is a sequence of characters. A null-terminated character array (a *string*) is a sequence of characters, the last of which is the null character. The null string is a character array containing only the terminating null character. A **NULL** pointer is the value that is obtained by casting **0** into a pointer. C guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return **NULL** to indicate an error. The macro **NULL** is defined in `<stdio.h>`. Types of the form **size_t** are defined in the appropriate headers.

Libraries are classified into four categories which define the level of the libraries' ability to support threads.

The MT-Level category of the libraries in this section are shown on each man page under **MT-Level**. Pages containing routines that are of multiple or differing MT-Levels show this under the **NOTES** section.

Safe Safe is simply an attribute of code that can be called from a multithreaded application. It is a generic term used to differentiate between code that is unsafe.

Unsafe An unsafe library contains global and static data that is not protected. It is not safe to use unless the application arranges for only one thread at time to execute within the library. Unsafe libraries may contain routines that are safe; however, most of the library's routines are unsafe to call.

MT-Safe An MT-Safe library is fully prepared for multithreaded access. It protects its global and static data with locks, and can provide a reasonable amount of concurrency. Note that a library can be safe to use, but not MT-Safe. For example, surrounding an entire library with a monitor makes the library safe, but it supports no concurrency so it is not considered MT-Safe. An MT-Safe library must permit a reasonable amount of concurrency. (This definition's purpose is to give precision to what is meant when a library is described as safe. The definition of a "safe" library does not specify if the library supports concurrency. The MT-Safe definition makes it clear that the library is safe, and supports some concurrency. This clarifies the safe definition, which can mean anything from being single threaded to being any degree of multithreaded.)

Async-Signal-Safe

Async-Signal-Safe refers to particular library routines that can be safely called from a signal handler. A thread that is executing an Async-Signal-Safe routine will not deadlock with itself if interrupted by a signal. Signals are only a problem for MT-Safe routines that acquire locks.

Signals are disabled when locks are acquired in Async-Signal-Safe routines. This prevents a signal handler that might acquire the same lock from being

called.

The designation "Async-Safe" also indicates "Async-Signal-Safe."

The list of "Async-Signal-Safe" functions includes:

_exit	kill	tcflow
access	link	tcflush
aio_error	lseek	tcgetattr
aio_return	mkdir	tcgetpgrp
aio_suspend	mkfifo	tcsendbreak
alarm	open	tcsetattr
cfgetispeed	pathconf	tcsetpgrp
cfgetospeed	pause	thr_kill
cfsetispeed	pipe	thr_sigsetmask
cfsetospeed	read	time
chdir	rename	timer_getoverrun
chmod	rmdir	timer_gettime
chown	sem_post	timer_settime
clock_gettime	sema_post	times
close	setgid	umask
creat	setpgid	uname
dup	setsid	unlink
dup2	setuid	utime
execle	sigaction	wait
execve	sigaddset	waitpid
fcntl	sigdelset	write
fdatasync	sigemptyset	
fork	sigfillset	
fstat	sigismember	
fsync	sigpending	
getegid	sigprocmask	
geteuid	sigqueue	
getgid	sigsuspend	
getgroups	sleep	
getpgrp	stat	
getpid	sysconf	
getppid	tcdrain	
getuid	tcdrain	

MT-Safe with exceptions

See the **NOTES** sections of these pages for a description of the exceptions.

Safe with exceptions

See the **NOTES** sections of these pages for a description of the exceptions.

Fork1-Safe

A fork1-safe library releases the locks it had held whenever **fork1(2)** is called in a Solaris thread program, or **fork(2)** in a POSIX thread program. Calling **fork(2)** in a POSIX thread program has the same semantic as calling

fork1(2) in a Solaris thread program. All system calls, **libpthread**, and **libthread** are **Fork1-Safe**. Otherwise, you should handle the locking clean-up yourself (see **pthread_atfork(3T)**).

The following table contains reentrant counterparts for unsafe functions. This table is subject to change by SunSoft.

Reentrant functions for libc

Unsafe Function Reentrant counterpart

ctime	ctime_r
localtime	localtime_r
asctime	asctime_r
gmtime	gmtime_r
ctermid	ctermid_r
getlogin	getlogin_r
rand	rand_r
readdir	readdir_r
strtok	strtok_r
tmpnam	tmpnam_r

Cancel-Safety

If a multi-threaded application uses **pthread_cancel(3T)** to cancel (i.e., kill a thread), it is possible that the target thread is killed while holding a resource, such as a lock or allocated memory. If the thread has not installed the appropriate cancellation cleanup handlers to release the resources appropriately (see **pthread_cancel(3T)**), the application is "cancel-unsafe", i.e., it is not safe with respect to cancellation. This unsafety could result in deadlocks due to locks not released by a thread that gets cancelled, or resource leaks; for example, memory not being freed on thread cancellation. All applications that use **pthread_cancel(3T)** should ensure that they operate in a cancel-safe environment.

Libraries that have cancellation points and which acquire resources such as locks or allocate memory dynamically, also contribute to the cancel-unsafety of applications that are linked with these libraries. This introduces another level of safety for libraries in a multi-threaded program: cancel-safety.

There are two sub-categories of cancel-safety:

Deferred-cancel-safety and Asynchronous-cancel-safety.

An application is considered to be Deferred-cancel-safe when it is cancel-safe for threads whose cancellation type is **PTHREAD_CANCEL_DEFERRED**.

An application is considered to be Asynchronous-cancel-safe when it is cancel-safe for threads whose cancellation type is **PTHREAD_CANCEL_ASYNCCHRONOUS**.

Deferred-cancel-safety is easier to achieve than Asynchronous-cancel-safety,

since a thread with the deferred cancellation type can be cancelled only at well-defined "cancellation points", whereas a thread with the asynchronous cancellation type can be cancelled anywhere. Since all threads are created by default to have the deferred cancellation type, it may never be necessary to worry about asynchronous cancel safety. Indeed, most applications and libraries are expected to always be Asynchronous-cancel-unsafe.

An application which is Asynchronous-cancel-safe is also, by definition, Deferred-cancel-safe.

FILES

<i>INCDIR</i>	usually /usr/include
<i>LIBDIR</i>	usually /usr/ccs/lib
<i>LIBDIR/libc.so</i>	
<i>LIBDIR/libc.a</i>	
<i>LIBDIR/libgen.a</i>	
<i>LIBDIR/libm.a</i>	
<i>LIBDIR/libsfm.sa</i>	
<i>/usr/lib/libc.so.1</i>	

SEE ALSO **ar(1)**, **cc(1B)**, **ld(1TSOL)**, **nm(1)**, **fork(2)**, **intro(2TSOL)**, **stdio(3S)**, **pthread_atfork(3T)**, **xpg4(5)**

ANSI C Programmer's Guide and the *Trusted Solaris Developer's Guide*

DIAGNOSTICS

For functions that return floating-point values, error handling varies according to compilation mode. Under the **-Xt** (default) option to **cc**, these functions return the conventional values **0**, **±HUGE**, or **NaN** when the function is undefined for the given arguments or when the value is not representable. In the **-Xa** and **-Xc** compilation modes, **±HUGE_VAL** is returned instead of **±HUGE**. (**HUGE_VAL** and **HUGE** are defined in **math.h** to be infinity and the largest-magnitude single-precision number, respectively.)

NOTES ON MULTITHREAD APPLICATIONS

When compiling a multithreaded application, either the **_POSIX_C_SOURCE**, **_POSIX_PTHREAD_SEMANTICS**, or **_REENTRANT** flag must be defined on the command line. This enables special definitions for functions only applicable to multithreaded applications. For POSIX.1c compliant applications, define the **_POSIX_C_SOURCE** flag to be **>= 199506L**:

```
cc [flags] file... -D_POSIX_C_SOURCE=199506L -lpthread
```

For POSIX behavior with the Solaris **fork()** and **fork1()** distinction, compile as follows:

```
cc [flags] file... -D_POSIX_PTHREAD_SEMANTICS -lpthread
```

For Solaris behavior, compile as follows:

```
cc [flags] file... -D_REENTRANT -lpthread
```

When building a singlethreaded application, the above flags should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.

When linking, it is a requirement that a multithreaded application be constructed to ensure that **libthread** physically interposes upon the C library. For example, the command line for linking the application using `.B ld` should be ordered as follows:

example% ld [options] .o's ... -lthread

Note that the behavior of the C library is undefined if `-lc` precedes `-lthread`. And, when linking with `cc`, `-lthread` or `-lpthread` must be last on the command line.

Unsafe interfaces should be called only from the main thread to ensure the application's safety.

MT-Safe interfaces are denoted in the NOTES section of the functions and libraries man pages. If a man page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.

REALTIME APPLICATIONS

NOTES

Be sure to have set the environment variable `LD_BIND_NOW` to a non-**NULL** value to enable early binding.

None of the functions, external variables, or macros should be redefined in the user's programs. Any other name may be redefined without affecting the behavior of other library functions, but such redefinition may conflict with a declaration in an included header.

The headers in *INCDIR* provide function prototypes (function declarations including the types of arguments) for most of the functions listed in this manual. Function prototypes allow the compiler to check for correct usage of these functions in the user's program. The **lint** program checker may also be used and will report discrepancies even if the headers are not included with **#include** statements. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the `-I` option to **lint**. (For example, `-lm` includes definitions for **libm**.) Use of **lint** is highly recommended. See the **lint** chapter in *Profiling Tools*.

Users should carefully note the difference between STREAMS and *stream*. STREAMS is a set of kernel mechanisms that support the development of network services and data communication drivers. It is composed of utility routines, kernel facilities, and a set of data structures. A *stream* is a file with its associated buffering. It is declared to be a pointer to a type **FILE** defined in `<stdio.h>`.

In detailed definitions of components, it is sometimes necessary to refer to symbolic names that are implementation-specific, but which are not necessarily expected to be accessible to an application program. Many of these symbolic names describe boundary conditions and system limits.

In this section, for readability, these implementation-specific values are given symbolic names. These names always appear enclosed in curly brackets to distinguish them from symbolic names of other implementation-specific constants that are accessible to application programs by headers. These names are not necessarily accessible to an application program through a header, although they may be defined in the documentation for a particular system.

In general, a portable application program should not refer to these symbolic names in its code. For example, an application program would not be expected to test the length of an argument list given to a routine to determine if it was greater than **{ARG_MAX}**.

Name	Description
__nis_map_group (3NTSOL)	See nis_groups (3NTSOL)
accept (3TSOL)	accept a connection on a socket
adornfc (3TSOL)	adorn the final component of a path-name
au_preselect (3TSOL)	preselect an audit event
au_user_mask (3TSOL)	get user's binary preselection mask
auditwrite (3TSOL)	construct and write user level audit records
auth_set_to_str (3TSOL)	See auth_to_str (3TSOL)
auth_to_str (3TSOL)	manipulate or verify authorizations
aw_errno (3TSOL)	See aw_strerror (3TSOL)
aw_perror (3TSOL)	See aw_strerror (3TSOL)
aw_strerror (3TSOL)	obtain and display auditwrite error messages
bclearhigh (3TSOL)	See blmanifest (3TSOL)
bclearlow (3TSOL)	See blmanifest (3TSOL)
bcleartoh (3TSOL)	See btohex (3TSOL)
bcleartoh_r (3TSOL)	See btohex (3TSOL)
bcleartos (3TSOL)	See bltos (3TSOL)
bclearundef (3TSOL)	See blmanifest (3TSOL)
bclearvalid (3TSOL)	See blvalid (3TSOL)
bclhigh (3TSOL)	See blmanifest (3TSOL)
bcllow (3TSOL)	See blmanifest (3TSOL)
bcltobanner (3TSOL)	translate binary CMW labels to character-coded labels for a printer banner
bcltoh (3TSOL)	See btohex (3TSOL)
bcltoh_r (3TSOL)	See btohex (3TSOL)
bcltoil (3TSOL)	See blportion (3TSOL)
bcltos (3TSOL)	See bltos (3TSOL)
bcltosl (3TSOL)	See blportion (3TSOL)

bclundef (3TSOL)	See blmanifest (3TSOL)
bilconjoin (3TSOL)	conjoin binary information labels
bildominates (3TSOL)	See blcompare (3TSOL)
bilequal (3TSOL)	See blcompare (3TSOL)
bilhigh (3TSOL)	See blmanifest (3TSOL)
billow (3TSOL)	See blmanifest (3TSOL)
biltoh (3TSOL)	See tohex (3TSOL)
biltoh_r (3TSOL)	See btohex (3TSOL)
biltolev (3TSOL)	See blportion (3TSOL)
biltos (3TSOL)	See bltos (3TSOL)
bilundef (3TSOL)	See blmanifest (3TSOL)
bilvalid (3TSOL)	See blvalid (3TSOL)
bimdominates (3TSOL)	See blcompare (3TSOL)
bimequal (3TSOL)	See blcompare (3TSOL)
bind (3NTSOL)	bind a name to a socket
blcompare (3TSOL)	compare binary labels
bldominates (3TSOL)	See blcompare (3TSOL)
blequal (3TSOL)	See blcompare (3TSOL)
blinrange (3TSOL)	See blcompare (3TSOL)
blinset (3TSOL)	check binary label for inclusion in set
blmanifest (3TSOL)	create manifest binary labels
blmaximum (3TSOL)	See blminmax (3TSOL)
blminimum (3TSOL)	See blminmax (3TSOL)
blminmax (3TSOL)	bound of two binary levels
blportion (3TSOL)	access binary label portions
blstrictdom (3TSOL)	See blcompare (3TSOL)
bltocolor (3TSOL)	get ASCII color name of label
bltocolor_r (3TSOL)	See bltocolor (3TSOL)
bltos (3TSOL)	translate binary labels to character-coded labels
bltype (3TSOL)	compare and set the type of binary label
blvalid (3TSOL)	check validity of binary label
bslhigh (3TSOL)	See blmanifest (3TSOL)
bsllow (3TSOL)	See blmanifest (3TSOL)
bsltoh (3TSOL)	See btohex (3TSOL)

bsltoh_r (3TSOL)	See btohex (3TSOL)
bsltos (3TSOL)	See bltos (3TSOL)
bslundef (3TSOL)	See blmanifest (3TSOL)
bslvalid (3TSOL)	See bslvalid (3TSOL)
btohex (3TSOL)	convert binary label to hexadecimal
chkauth (3TSOL)	See auth_to_str (3TSOL)
clnt_call (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_control (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_create_timed (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_create_vers (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_destroy (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_dg_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_freeres (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_geterr (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_pcreateerror (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_perrno (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_perror (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_raw_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_spcreateerror (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_sperrno (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_sperror (3NTSOL)	See rpc_clnt_calls (3NTSOL)
clnt_tli_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_tp_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_tp_create_timed (3NTSOL)	See rpc_clnt_create (3NTSOL)
clnt_vc_create (3NTSOL)	See rpc_clnt_create (3NTSOL)
clock_getres (3RTSOL)	See clock_settime (3RTSOL)
clock_gettime (3RTSOL)	See clock_settime (3RTSOL)
clock_settime (3RTSOL)	high-resolution clock operations
dn_comp (3NTSOL)	See resolver (3NTSOL)
dn_expand (3NTSOL)	See resolver (3NTSOL)
endac (3TSOL)	See getacinfo (3TSOL)
endauclass (3TSOL)	See getauclassent (3TSOL)
endauevent (3TSOL)	See getauevent (3TSOL)
endauser (3TSOL)	See getprofent (3TSOL)

endprofent (3TSOL)	See getprofent (3TSOL)
endprofstr (3TSOL)	See getprofstr (3TSOL)
enduserent (3TSOL)	See getuserent (3TSOL)
endutent (3CTSOL)	See getutent (3CTSOL)
endutxent (3CTSOL)	See getutxent (3CTSOL)
free_auth_set (3TSOL)	See auth_to_str (3TSOL)
free_profent (3TSOL)	See getprofent (3TSOL)
free_profstr (3TSOL)	See get_profstr (3TSOL)
free_userent (3TSOL)	See getuserent (3TSOL)
ftw (3CTSOL)	walk a file tree
get_auth_text (3TSOL)	See auth_to_str (3TSOL)
get_priv_text (3TSOL)	See priv_to_str (3TSOL)
getacdir (3TSOL)	See getacinfo (3TSOL)
getacflg (3TSOL)	See getacinfo (3TSOL)
getacinfo (3TSOL)	get audit control file information
getacmin (3TSOL)	See getacinfo (3TSOL)
getacna (3TSOL)	See getacinfo (3TSOL)
getaclassent (3TSOL)	get audit_class entry
getaclassent_r (3TSOL)	See getaclassent (3TSOL)
getaclassnam (3TSOL)	See getaclassent (3TSOL)
getaclassnam_r (3TSOL)	See getaclassent (3TSOL)
getauditflags (3TSOL)	convert audit flag specifications
getauditflagsbin (3TSOL)	See getauditflags (3TSOL)
getauditflagschar (3TSOL)	See getauditflags (3TSOL)
getauevent (3TSOL)	get audit_event entry
getauevent_r (3TSOL)	See getauevent (3TSOL)
getauevnam (3TSOL)	See getauevent (3TSOL)
getauevnam_r (3TSOL)	See getauevent (3TSOL)
getauevnonam (3TSOL)	See getauevent (3TSOL)
getauevnum (3TSOL)	See getauevent (3TSOL)
getauevnum_r (3TSOL)	See getauevent (3TSOL)
get_auth_text (3TSOL)	See auth_to_str (3TSOL)
getauuserent (3TSOL)	See getauusernam (3TSOL)
getauuserent_r (3TSOL)	See getauusernam (3TSOL)
getauusernam (3TSOL)	get audit_user entry

getauusernam_r (3TSOL)	See getauusernam (3TSOL)
getcil (3TSOL)	See blportion (3TSOL)
getcsl (3TSOL)	See blportion (3TSOL)
getfauditflags (3TSOL)	generate the process audit state
getpeerinfo (3TSOL)	get peer process' characteristics
getprofent (3TSOL)	get Trusted Solaris user profile description
getprofentbyname (3TSOL)	See getprofent (3TSOL)
getprofstr (3TSOL)	get Trusted Solaris user profile description
getprofstrbyname (3TSOL)	See getprofstr (3TSOL)
getsockopt (3NTSOL)	get and set options on sockets
getuserent (3TSOL)	get Trusted Solaris user security attributes
getuserentbyname (3TSOL)	See getuserent (3TSOL)
getuserentbyuid (3TSOL)	See getuserent (3TSOL)
getutent (3CTSOL)	access utmp file entry
getutid (3CTSOL)	See getutent (3CTSOL)
getutline (3CTSOL)	See getutent (3CTSOL)
getutmp (3CTSOL)	See getutxent (3CTSOL)
getutmpx (3CTSOL)	See getutxent (3CTSOL)
getutxent (3CTSOL)	access utmpx file entry
getutxid (3CTSOL)	See getutxent (3CTSOL)
getutxline (3CTSOL)	See getutxent (3CTSOL)
getvfsaent (3TSOL)	get vfstab_adjunct file entry
getvfssafe (3TSOL)	See getvfsaent (3TSOL)
h_alloc (3TSOL)	See btohex (3TSOL)
h_free (3TSOL)	See btohex (3TSOL)
hextob (3TSOL)	convert hexadecimal string to binary label
htobcl (3TSOL)	See hextob (3TSOL)
htobclear (3TSOL)	See hextob (3TSOL)
htobil (3TSOL)	See hextob (3TSOL)
htobsl (3TSOL)	See hextob (3TSOL)
initgroups (3CTSOL)	initialize the supplementary group access list

kstat_read (3KTSOL)	read or write kstat data
kstat_write (3KTSOL)	See kstat_read (3KTSOL)
labelbuilder (3TSOL)	create a Motif-based user interface for interactively building a valid label or clearance
labelclipping (3TSOL)	translate a binary label and clip to specified width
labelinfo (3TSOL)	get information about the label encodings
labelvers (3TSOL)	get version of the label_encodings file
libt6 (3NTSOL)	TSIX trusted IPC library
listen (3NTSOL)	listen for connections on a socket
mldgetcwd (3TSOL)	get pathname of current working directory
mldrealpath (3TSOL)	return the pathname, including adornments and SLD names
mldrealpathl (3TSOL)	See mldrealpath (3TSOL)
mlock (3CTSOL)	lock (or unlock) pages in memory
mlockall (3CTSOL)	lock or unlock address space
munlock (3CTSOL)	See mlock (3CTSOL)
munlockall (3CTSOL)	See mlockall (3CTSOL)
nftw (3CTSOL)	See ftw (3CTSOL)
nis_add (3NTSOL)	See nis_names (3NTSOL)
nis_add_entry (3NTSOL)	See nis_tables (3NTSOL)
nis_addmember (3NTSOL)	See nis_groups (3NTSOL)
nis_checkpoint (3NTSOL)	See nis_ping (3NTSOL)
nis_creategroup (3NTSOL)	See nis_groups (3NTSOL)
nis_destroygroup (3NTSOL)	See nis_groups (3NTSOL)
nis_first_entry (3NTSOL)	See nis_tables (3NTSOL)
nis_freeresult (3NTSOL)	See nis_names (3NTSOL)
nis_freeservlist (3NTSOL)	See nis_server (3NTSOL)
nis_freetags (3NTSOL)	See nis_server (3NTSOL)
nis_getservlist (3NTSOL)	See nis_server (3NTSOL)
nis_groups (3NTSOL)	NIS+ group manipulation functions
nis_ismember (3NTSOL)	See nis_groups (3NTSOL)
nis_list (3NTSOL)	See nis_tables (3NTSOL)

nis_lookup (3NTSOL)	See nis_names (3NTSOL)
nis_map_group (3NTSOL)	See nis_groups (3NTSOL)
nis_mkdir (3NTSOL)	See nis_server (3NTSOL)
nis_modify (3NTSOL)	See nis_names (3NTSOL)
nis_modify_entry (3NTSOL)	See nis_tables (3NTSOL)
nis_names (3NTSOL)	NIS+ namespace functions
nis_next_entry (3NTSOL)	See nis_tables (3NTSOL)
nis_ping (3NTSOL)	misc NIS+ log administration functions
nis_print_group_entry (3NTSOL)	See nis_groups (3NTSOL)
nis_remove (3NTSOL)	See nis_names (3NTSOL)
nis_remove_entry (3NTSOL)	See nis_tables (3NTSOL)
nis_removemember (3NTSOL)	See nis_groups (3NTSOL)
nis_rmdir (3NTSOL)	See nis_server (3NTSOL)
nis_server (3NTSOL)	miscellaneous NIS+ functions
nis_servstate (3NTSOL)	See nis_server (3NTSOL)
nis_stats (3NTSOL)	See nis_server (3NTSOL)
nis_tables (3NTSOL)	NIS+ table functions
nis_verifygroup (3NTSOL)	See nis_groups (3NTSOL)
plock (3CTSOL)	lock or unlock into memory process, text, or data
priv_set_to_str (3TSOL)	See priv_to_str (3TSOL)
priv_to_str (3TSOL)	convert privilege IDs and names
putprofstr (3TSOL)	See getprofstr (3TSOL)
pututline (3CTSOL)	See getutent (3CTSOL)
pututxline (3CTSOL)	See getutxent (3CTSOL)
randomword (3TSOL)	generate random pronounceable password
res_init (3NTSOL)	See resolver (3NTSOL)
res_mkquery (3NTSOL)	See resolver (3NTSOL)
res_search (3NTSOL)	See resolver (3NTSOL)
res_send (3NTSOL)	See resolver (3NTSOL)
resolver (3NTSOL)	resolver routines
rpc (3NTSOL)	library routines for remote procedure calls
rpc_broadcast (3NTSOL)	See rpc_clnt_calls (3NTSOL)

rpc_broadcast_exp (3NTSOL)	See rpc_clnt_calls (3NTSOL)
rpc_call (3NTSOL)	See rpc_clnt_calls (3NTSOL)
rpc_clnt_calls (3NTSOL)	library routines for client side calls
rpc_clnt_create (3NTSOL)	library routines for dealing with creation and manipulation of CLIENT handles
rpc_createerr (3NTSOL)	See rpc_clnt_create (3NTSOL)
rpc_reg (3NTSOL)	See rpc_svc_reg (3NTSOL)
rpc_svc_calls (3NTSOL)	library routines for RPC servers
rpc_svc_create (3NTSOL)	library routines for the creation of server handles
rpc_svc_reg (3NTSOL)	library routines for registering servers
rpcb_getmaps (3NTSOL)	See rpcbind (3NTSOL)
rpcb_getallmaps (3NTSOL)	See rpcbind (3NTSOL)
rpcb_getaddr (3NTSOL)	See rpcbind (3NTSOL)
rpcb_gettime (3NTSOL)	See rpcbind (3NTSOL)
rpcb_rmtcall (3NTSOL)	See rpcbind (3NTSOL)
rpcb_set (3NTSOL)	See rpcbind (3NTSOL)
rpcb_unset (3NTSOL)	See rpcbind (3NTSOL)
rpcbind (3NTSOL)	library routines for RPC bind service
sbcleartos (3TSOL)	See sbltos (3TSOL)
sbcltos (3TSOL)	See sbltos (3TSOL)
sbiltos (3TSOL)	See sbltos (3TSOL)
sbltos (3TSOL)	translate binary labels to canonical character-coded labels
sbsltos (3TSOL)	See sbltos (3TSOL)
send (3NTSOL)	send a message from a socket
sendmsg (3NTSOL)	See send (3NTSOL)
sendto (3NTSOL)	See send (3NTSOL)
set_effective_priv (3TSOL)	assign a privilege set for the current process
set_inheritable_priv (3TSOL)	See set_effective_priv (3TSOL)
set_permitted_priv (3TSOL)	See set_effective_priv (3TSOL)
setac (3TSOL)	See getacinfo (3TSOL)
setauclass (3TSOL)	See getauclassent (3TSOL)

setauevent (3TSOL)	See getauevent (3TSOL)
setauuser (3TSOL)	See getauusernam (3TSOL)
setbltype (3TSOL)	See bltype (3TSOL)
setcil (3TSOL)	See blportion (3TSOL)
setcsl (3TSOL)	See blportion (3TSOL)
setprofent (3TSOL)	See getprofent (3TSOL)
setprofstr (3TSOL)	See getprofstr (3TSOL)
setsockopt (3NTSOL)	See getsockopt (3NTSOL)
setuserent (3TSOL)	See getuserent (3TSOL)
setutent (3CTSOL)	See getutent (3CTSOL)
setutxent (3CTSOL)	See getutxent (3CTSOL)
stobl (3TSOL)	See stobl (3TSOL)
stobclear (3TSOL)	See stobl (3TSOL)
stobil (3TSOL)	ee stobl (3TSOL)
stobl (3TSOL)	translate character-coded labels to binary labels
stobsl (3TSOL)	See stobl (3TSOL)
str_to_auth (3TSOL)	See auth_to_str (3TSOL)
str_to_auth_set (3TSOL)	See auth_to_str (3TSOL)
str_to_priv (3TSOL)	See priv_to_str (3TSOL)
str_to_priv_set (3TSOL)	See priv_to_str (3TSOL)
svc_auth_reg (3NTSOL)	See rpc_svc_reg (3NTSOL)
svc_control (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_destroy (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_dg_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_dg_enablecache (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_done (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_exit (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_fd_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_fdset (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_freeargs (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_getargs (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_getreq_common (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_getreq_poll (3NTSOL)	See rpc_svc_calls (3NTSOL)

svc_getreqset (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_getrpccaller (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_pollset (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_raw_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_reg (3NTSOL)	See rpc_svc_reg (3NTSOL)
svc_run (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_sendreply (3NTSOL)	See rpc_svc_calls (3NTSOL)
svc_tli_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_tp_create (3NTSOL)	See rpc_svc_create (3NTSOL)
svc_unreg (3NTSOL)	See rpc_svc_reg (3NTSOL)
svc_vc_create (3NTSOL)	See rpc_svc_create (3NTSOL)
t6alloc_blk (3NTSOL)	allocate and free security attribute control
t6attr_query (3NTSOL)	get mask indicating which attributes came from templates
t6clear_blk (3NTSOL)	clear security attributes
t6cmp_blk (3NTSOL)	compare security attributes
t6copy_blk (3NTSOL)	copy security attributes
t6dup_blk (3NTSOL)	duplicate security attributes
t6ext_attr (3NTSOL)	manipulate network endpoint security options
t6free_blk (3NTSOL)	See t6alloc_blk (3NTSOL)
t6get_attr (3NTSOL)	get and set security attributes from / to the security attribute buffer
t6get_endpt_default (3NTSOL)	See t6get_endpt_mask (3NTSOL)
t6get_endpt_mask (3NTSOL)	get and set endpoint mask, or get and set endpoint default attributes
t6last_attr (3NTSOL)	See t6peek_attr (3NTSOL)
t6new_attr (3NTSOL)	See t6ext_attr (3NTSOL)
t6peek_attr (3NTSOL)	examine the security attributes on the next / previous byte
t6recvfrom (3NTSOL)	read security attributes and data from a trusted endpoint
t6sendto (3NTSOL)	specify security attributes to send with data on a trusted endpoint
t6set_attr (3NTSOL)	See t6get_attr (3NTSOL)
t6set_endpt_default (3NTSOL)	See t6get_endpt_mask (3NTSOL)

t6set_endpt_mask (3NTSOL)	See t6get_endpt_mask (3NTSOL)
t6size_attr (3NTSOL)	get the size of a particular attribute from the control structure
t_accept (3NTSOL)	accept a connect request
t_bind (3NTSOL)	bind an address to a transport endpoint
t_optmgmt (3NTSOL)	manage options for a transport endpoint
t_snd (3NTSOL)	send data or expedited data over a connection
t_sndudata (3NTSOL)	send a data unit
tsol_lbuild_create (3TSOL)	See labelbuilder (3TSOL)
tsol_lbuild_destroy (3TSOL)	See labelbuilder (3TSOL)
tsol_lbuild_get (3TSOL)	See labelbuilder (3TSOL)
tsol_lbuild_set (3TSOL)	See labelbuilder (3TSOL)
updwtmp (3CTSOL)	See getutxent (3CTSOL)
updwtmpx (3CTSOL)	See getutxent (3CTSOL)
utmpname (3CTSOL)	See getutent (3CTSOL)
utmpxname (3CTSOL)	See getutxent (3CTSOL)
xprt_register (3NTSOL)	See rpc_svc_reg (3NTSOL)
xprt_unregister (3NTSOL)	See rpc_svc_reg (3NTSOL)
Xbclear (3TSOL)	See labelclipping (3TSOL)
Xbcltos (3TSOL)	See labelclipping (3TSOL)
Xbiltos (3TSOL)	See labelclipping (3TSOL)
Xbsltos (3TSOL)	See labelclipping (3TSOL)
XTSOLIsWindowTrusted (3X11TSOL)	test if a window is created by a trusted client
XTSOLMakeTPWindow (3X11TSOL)	make this window a Trusted Path window
XTSOLgetClientAttributes (3X11TSOL)	get all CMW attributes associated with a client
XTSOLgetPropAttributes (3X11TSOL)	get all CMW attributes associated with a property hanging on a window
XTSOLgetPropLabel (3X11TSOL)	get the CMW label associated with a property hanging on a window
XTSOLgetPropUID (3X11TSOL)	get the UID associated with a property hanging on a window

XTSOLgetResAttributes (3X11TSOL)	get all CMW attributes associated with a window or a pixmap
XTSOLgetResLabel (3X11TSOL)	get the CMW label associated with a window, a pixmap, or a colormap
XTSOLgetResUID (3X11TSOL)	get the UID associated with a window, a pixmap
XTSOLgetSSHeight (3X11TSOL)	get the height of screen stripe
XTSOLgetWindowIIL (3X11TSOL)	Get the input information label of a window
XTSOLgetWorkStationOwner (3X11TSOL)	get the ownership of the workstation
XTSOLShutdown (3X11TSOL)	shut down the system
XTSOLsetPolyInstInfo (3X11TSOL)	set polyinstantiation information
XTSOLsetPropLabel (3X11TSOL)	set the CMW label associated with a property hanging on a window
XTSOLsetPropUID (3X11TSOL)	set the UID associated with a property hanging on a window
XTSOLsetResLabel (3X11TSOL)	set the CMW label associated with a window or a pixmap
XTSOLsetResUID (3X11TSOL)	set the UID associated with a window, a pixmap, or a colormap
XTSOLsetSSHeight (3X11TSOL)	set the height of the screenstripe
XTSOLsetSessionHI (3X11TSOL)	set the session high sensitivity label to the window server
XTSOLsetSessionLO (3X11TSOL)	set the session low sensitivity label to the window server
XTSOLsetWindowIIL (3X11TSOL)	set the input information label of a window
XTSOLsetWorkStationOwner (3X11TSOL)	set the ownership of the workstation

NAME	XTSOLIsWindowTrusted – Test if a window is created by a trusted client
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLIsWindowTrusted (<i>display</i> , <i>window</i>) Display <i>*display</i> ; Window <i>window</i> ;
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLIsWindowTrusted() tests if a window is created by a trusted client. The window created by a trusted client has a special bit turned on. The client does not require any privilege to perform this operation.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay . <i>window</i> Specifies the ID of the window to be tested.
RETURN VALUES	True if the window is created by a trusted client.
ERRORS	BadWindow Not a valid window

NAME	XTSOLMakeTPWindow – Make this window a Trusted Path window	
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLMakeTPWindow (<i>display</i> , <i>w</i>) Display <i>*display</i> ; Window <i>w</i> ;	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLMakeTPWindow() is used to make a window a trusted path window. Trusted Path windows always remain on top of other windows. The client must have the Trusted Path process attribute set.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>w</i>	Specifies the ID of a window.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadValue	Not a valid type

NAME	XTSOLShutdown – Shut down the system
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLShutdown (<i>display</i>) Display * <i>display</i> ;
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLShutdown() shuts down the system of <i>display</i> . The client must have the PRIV_SYS_BOOT privilege to perform this operation.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay .

NAME	XTSOLgetClientAttributes – Get all CMW attributes associated with a client
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLgetClientAttributes (<i>display</i>, <i>windowid</i>, <i>clientattrp</i>) Display *<i>display</i>; XID <i>windowid</i>; XTSOLClientAttributes *<i>clientattrp</i>;</pre>
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLgetClientAttributes() is used to get all CMW attributes associated with a client in a single call. The attributes include process ID, user ID, IP address, audit flags and session ID.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay. <i>windowid</i> Specifies window ID of X client. <i>clientattrp</i> Client must provide a pointer to an XTSOLClientAttributes structure.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege BadValue Not a valid client
SEE ALSO	XTSOLgetResAttributes(3X11TSOL) , XTSOLgetPropertyAttributes(3X11TSOL)

NAME	XTSOLgetPropAttributes – Get all CMW attributes associated with a property hanging on a window	
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetPropAttributes (<i>display</i>, <i>window</i>, <i>property</i>, <i>cmwpropattrp</i>) Display <i>*display;</i> Window <i>window;</i> Atom <i>property;</i> XTSOLPropAttributes <i>*cmwpropattrp;</i>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropAttributes() is used to get all CMW attributes associated with a property hanging out of a window in a single call. The attributes include UID, information label, and sensitivity label.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpen-Display .
	<i>window</i>	Specifies the ID of a window system object.
	<i>property</i>	Specifies the property atom.
	<i>cmwwinattrp</i>	Client must provide a pointer to XTSOLPropAttributes .
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadAtom	Not a valid atom
SEE ALSO	XTSOLgetResAttributes(3X11TSOL) , XTSOLgetClientAttributes(3X11TSOL)	

NAME	XTSOLgetPropLabel – Get the CMW label associated with a property hanging on a window
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLgetPropLabel (display, window, property, cmwlabel) Display *display; Window window; Atom property; bclabel_t *cmwlabel;</pre>
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	Client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropLabel() is used to get the CMW label associated with a property hanging on a window.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay. <i>window</i> Specifies the ID of the window whose property's CMW label you want to get. <i>property</i> Specifies the property atom. <i>cmwlabel</i> Returns a CMW label that is the current CMW label of the specified property. This label contains both SL and IL. Client needs to provide a bclabel_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to bclabel_t.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege BadWindow Not a valid window BadAtom Not a valid atom
SEE ALSO	XTSOLsetPropLabel(3X11TSOL) , XTSOLgetPropAttributes(3X11TSOL)

NAME	XTSOLgetPropUID – Get the UID associated with a property hanging on a window
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLgetPropUID (display, window, property, uidp) Display *display; Window window; Atom property; uid_t *uidp;</pre>
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropUID() gets the ownership of a window's property. This allows a client to get the ownership of an object it did not create.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay. <i>window</i> Specifies the ID of the window whose property's UID you want to get. <i>property</i> Specifies the property atom. <i>uidp</i> Returns a UID which is the current UID of the specified property. Client needs to provide a uid_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to uid_t.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege BadWindow Not a valid window BadAtom Not a valid atom
SEE ALSO	XTSOLsetPropUID(3X11TSOL) , XTSOLgetPropAttributes(3X11TSOL) .

NAME	XTSOLgetResAttributes – Get all CMW attributes associated with a window or a pixmap	
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetResAttributes (<i>display, object, type, cmwwinattrp</i>) Display <i>*display;</i> XID <i>object;</i> ResourceType <i>type;</i> XTSOLResAttributes <i>*cmwwinattrp;</i>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetResAttributes() is used to get all CMW attributes associated with a window or a pixmap in a single call. The attributes include UID, information label, sensitivity label, input information label, and workstation owner.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpen-Display .
	<i>object</i>	Specifies the ID of a window system object. Possible window system objects are windows and pixmaps.
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap
	<i>cmwwinattrp</i>	Client must provide a pointer to XTSOLResAttributes .
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadPixmap	Not a valid pixmap
	BadValue	Not a valid type
SEE ALSO	XTSOLgetPropAttributes(3X11TSOL) , XTSOLgetClientAttributes(3X11TSOL)	

NAME	XTSOLgetResLabel – Get the CMW label associated with a window, a pixmap, or a colormap
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLgetResLabel (display, object, type, cmwlabel) Display *display; XID object; ResourceType type; bclabel_t *cmwlabel;</pre>
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetResLabel() is used to get the CMW label associated with a window or a pixmap or a colormap.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay. <i>object</i> Specifies the ID of a window system object whose CMW label you want to get. Possible window system objects are windows, and pixmaps or colormaps. <i>type</i> Specifies what type of resource is being accessed. Possible values are IsWindow, IsPixmap or IsColormap. <i>cmwlabel</i> Returns a CMW label which is the current CMW label of the specified object. This label contains both SL and IL. Client needs to provide a bclabel_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to bclabel_t.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege BadPixmap Not a valid pixmap BadValue Not a valid type
SEE ALSO	XTSOLsetResLabel(3X11TSOL) , XTSOLgetClientAttributes(3X11TSOL)

NAME	XTSOLgetResUID – Get the UID associated with a window, a pixmap	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLgetResUID (display, object, type, uidp) Display *display; XID object; ResourceType type; uid_t *uidp;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetResUID() gets the ownership of a window system object. This allows a client to get the ownership of an object it did not create.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>object</i>	Specifies the ID of a window system object whose UID you want to get. Possible window system objects are windows or pixmaps.
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap .
	<i>uidp</i>	Returns a UID which is the current UID of the specified object. Client must provide a pointer to uid_t .
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadPixmap	Not a valid pixmap
	BadValue	Not a valid type
SEE ALSO	XTSOLgetResUID(3X11TSOL) , XTSOLgetClientAttributes(3X11TSOL) , XTSOLgetResAttributes(3X11TSOL) .	

NAME	XTSOLgetSSHHeight – Get the height of screen stripe.	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLgetSSHHeight (<i>display</i>, <i>screen_num</i>, <i>newheight</i>) Display *<i>display</i>; int <i>screen_num</i>; int * <i>newheight</i>;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLgetSSHHeight() gets the height of trusted screen stripe at the bottom of the screen. Currently the screen stripe is only present on the default screen. Client must have the Trusted Path process attribute.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpen-Display .
	<i>screen_num</i>	Specifies the screen number.
	<i>newheight</i>	Specifies the storage area where the height of the stripe in pixels is returned.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i>
SEE ALSO	XTSOLsetSSHHeight(3X11TSOL)	

NAME	XTSOLgetWindowIIL – Get the input information label of a window	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLgetWindowIIL (<i>display</i>, <i>window</i>, <i>iilabel</i>) Display *<i>display</i>; Window <i>window</i>; bilabel_t *<i>iilabel</i>;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLgetWindowIIL() returns the IIL associated with a window. The IIL is associated with the client's top-level window. Subwindows created under a client's top-level window all have the same IIL. The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>window</i>	Specifies the ID of the window whose IIL you want to get.
	<i>iilabel</i>	Returns an input information label which is the current IIL of the specified window. Client must provide a pointer to bilabel_t .
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
SEE ALSO	XTSOLsetWindowIIL (3X11TSOL)	

NAME	XTSOLgetWorkstationOwner – Get the ownership of the workstation
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLgetWorkstationOwner (<i>display</i> , <i>uidp</i>) Display <i>*display</i> ; uid_t <i>*uidp</i> ;
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLgetWorkstationOwner() is used to get the ownership of the workstation.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay . <i>uidp</i> Returns a UID which is the current UID of the specified Display workstation server. Client must provide a pointer to uid_t .
RETURN VALUES	None.
ERRORS	BadAccess Lack of privilege
SEE ALSO	XTSOLsetWorkstationOwner (3X11TSOL)

NAME	XTSOLsetPolyInstInfo – Set polyinstantiation information	
SYNOPSIS	<pre> #include <tsol/Xtsol.h> XTSOLsetPolyInstInfo (<i>display</i>, <i>sl</i>, <i>uidp</i>, <i>enabled</i>) Display *<i>display</i>; bslabel_t *<i>sl</i>; uid_t * <i>uidp</i>; int <i>enabled</i>; </pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLsetPolyInstInfo() sets the polyinstantiated information to get property resources. By default, when a client requests property data for a polyinstantiated property, the data returned corresponds to the SL and UID of the requesting client. To get the property data associated with a property with specific <i>sl</i> and <i>uid</i> a client can use this call to set the SL and UID with <i>enabled</i> flag to TRUE. The client should also restore the <i>enabled</i> flag to FALSE after retrieving the property value. Client must have the PRIV_WIN_MAC_WRITE and PRIV_WIN_DAC_WRITE privileges.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>sl</i>	Specifies the sensitivity label.
	<i>uidp</i>	Specifies the pointer to UID.
	<i>enabled</i>	Specifies whether client can set the property information retrieved.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadValue	Not a valid <i>display</i> or <i>sl</i> .

NAME	XTSOLsetPropLabel – Set the CMW label associated with a property hanging on a window	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetPropLabel (<i>display</i>, <i>window</i>, <i>property</i>, <i>cmwlabel</i>, <i>labelFlag</i>) Display *<i>display</i>; Window <i>window</i>; Atom <i>property</i>; bclabel_t *<i>cmwlabel</i>; enum setting_flag <i>flag</i>;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLsetPropLabel() is used to change the CMW label associated with a property hanging on a window. The client must have the PRIV_WIN_DAC_WRITE, PRIV_WIN_MAC_WRITE, PRIV_WIN_UPGRADE_SL, PRIV_WIN_UPGRADE_IL, PRIV_WIN_DOWNGRADE_SL, and PRIV_WIN_DOWNGRADE_IL privileges.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>window</i>	Specifies the ID of the window whose property's CMW label you want to change.
	<i>property</i>	Specifies the property atom.
	<i>cmwlabel</i>	Specifies a pointer to a CMW label structure which contains a CMW label. Only a portion (depends on <i>labelFlag</i>) of the CMW label needs to be specified. The unspecified portion of the CMW label is not interpreted by the server.
	<i>labelFlag</i>	Specifies which portion of the CMW Label will be changed. Possible values are: SETCL_ALL , SETCL_SL , and SETCL_IL .
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadAtom	Not a valid atom
	BadValue	Not a valid <i>labelFlag</i> or <i>cmwlabel</i>
SEE ALSO	XTSOLgetPropLabel(3X11TSOL) , XTSOLgetPropAttributes(3X11TSOL)	

NAME	XTSOLsetPropUID – Set the UID associated with a property hanging on a window
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetPropUID (<i>display</i>, <i>window</i>, <i>property</i>, <i>uidp</i>) Display *<i>display</i>; Window <i>window</i>; Atom <i>property</i>; uid_t *<i>uidp</i>;</pre>
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLsetPropUID() changes the ownership of a window's property. This allows another client to modify a property of a window that it did not create. The client must have the PRIV_WIN_DAC_WRITE and PRIV_WIN_MAC_WRITE privileges.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay. <i>window</i> Specifies the ID of the window whose property's UID you want to change. <i>property</i> Specifies the property atom. <i>uidp</i> Specifies a pointer to a uid_t that contains a UID.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege BadWindow Not a valid window BadAtom Not a valid atom
SEE ALSO	XTSOLgetPropUID(3X11TSOL) , XTSOLgetPropAttributes(3X11TSOL)

NAME	XTSOLsetResLabel – Set the CMW label associated with a window or a pixmap		
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetResLabel (<i>display</i> , <i>object</i> , <i>type</i> , <i>cmwlabel</i> , <i>labelFlag</i>) Display <i>*display</i> ; XID <i>object</i> ; ResourceType <i>type</i> ; bclabel_t <i>*cmwlabel</i> ; enum setting_flag <i>labelFlag</i> ;		
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)		
DESCRIPTION	The client must have the PRIV_WIN_DAC_WRITE , PRIV_WIN_MAC_WRITE , PRIV_WIN_UPGRADE_SL , PRIV_WIN_UPGRADE_IL , PRIV_WIN_DOWNGRADE_SL , and PRIV_WIN_DOWNGRADE_IL privileges. XTSOLsetResLabel() is used to change the CMW label associated with a window or a pixmap.		
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .	
	<i>object</i>	Specifies the ID of a window system object whose CMW label you want to change. Possible window system objects are windows and pixmaps. The CMW label is not interpreted by the server.	
	<i>labelFlag</i>	Specifies which portion of the CMW label will be changed. Possible values are: RES_ALL , RES_SL , RES_IL , and RES_IIL .	
RETURN VALUES	None		
ERRORS	BadAccess	Lack of privilege	
	BadPixmap	Not a valid pixmap	
	BadValue	Not a valid <i>type</i> , <i>labelFlag</i> , or <i>cmwlabel</i>	
SEE ALSO	XTSOLgetResLabel(3X11TSOL) , XTSOLgetResAttributes(3X11TSOL)		

NAME	XTSOLsetResUID – Set the UID associated with a window, a pixmap, or a colormap	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLsetResUID (<i>display</i>, <i>object</i>, <i>type</i>, <i>uidp</i>) Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; uid_t *<i>uidp</i>;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	The client must have the PRIV_WIN_DAC_WRITE and PRIV_WIN_MAC_WRITE privileges. XTSOLsetResUID() changes the ownership of a window system object. This allows a client to create an object and then change its ownership. The new owner can then make modifications on this object as this object being created by itself.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>object</i>	Specifies the ID of a window system object whose UID you want to change. Possible window system objects are windows and pixmaps.
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are: IsWindow and IsPixmap .
	<i>uidp</i>	Specifies a pointer to a uid_t structure that contains a UID.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadPixmap	Not a valid pixmap
	BadValue	Not a valid type
SEE ALSO	XTSOLgetResUID(3X11TSOL)	

NAME	XTSOLsetSSHheight – Set the height of screen stripe	
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetSSHheight (<i>display</i> , <i>screen_num</i> , <i>newheight</i>) Display <i>*display</i> ; int <i>screen_num</i> ; int <i>newheight</i> ;	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLsetSSHheight() sets the height of the trusted screen stripe at the bottom of the screen. Currently the screen stripe is present only on the default screen. The client must have the Trusted Path process attribute.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>screen_num</i>	Specifies the screen number.
	<i>newheight</i>	Specifies the height of the stripe in pixels.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i> .
SEE ALSO	XTSOLgetSSHheight(3X11TSOL)	

NAME	XTSOLsetSessionHI – Set the session high sensitivity label to the window server	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetSessionHI (display, sl) Display *display; blabel_t *sl;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLsetSessionHI() After the session high label has been set by a Trusted Solaris window system TCB component, logintool , Xsun will reject connection request from clients running at higher sensitivity labels than the session high label. The client must have the PRIV_WIN_CONFIG privilege.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>sl</i>	Specifies a pointer to a sensitivity label to be used as the session HIGH label.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
SEE ALSO	XTSOLsetSessionLO(3X11TSOL)	

NAME	XTSOLsetSessionLO – Set the session low sensitivity label to the window server		
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetSessionLO (display, sl) Display *display; bslabel_t *sl;		
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)		
MT-LEVEL	Unsafe		
DESCRIPTION	XTSOLsetSessionLO() sets the session low sensitivity label. After the session low label has been set by a Trusted Solaris window system TCB component, logintool , Xsun will reject a connection request from a client running at a lower sensitivity label than the session low label. The client must have the PRIV_WIN_CONFIG privilege.		
ARGUMENTS	display	Specifies a pointer to the Display structure; returned from XOpenDisplay .	
	sl	Specifies a pointer to a sensitivity label to be used as the session low label.	
RETURN VALUES	None		
ERRORS	BadAccess	Lack of privilege	
SEE ALSO	XTSOLsetSessionHI(3X11TSOL)		

NAME	XTSOLsetWindowIIL – Set the input information label of a window	
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetWindowIIL (<i>display</i>, <i>window</i>, <i>iilabel</i>) Display *<i>display</i>; Window <i>window</i>; bilabel_t <i>iilabel</i>;</pre>	
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)	
MT-LEVEL	Unsafe	
DESCRIPTION	XTSOLsetWindowIIL() changes the input information label associated with a window. Note that the specified window and all other windows in the same subtree are simultaneously affected. (The IIL is stored in the top-level-window.) The IIL is affected on a per client's top-level window basis; that is, subwindows created under a client's top-level window all have the same IIL. The client must have the PRIV_WIN_MAC_WRITE privilege.	
ARGUMENTS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay .
	<i>window</i>	Specifies the ID of the window.
	<i>iilabel</i>	Specifies a pointer to a bilabel_t structure that contains an input information label.
RETURN VALUES	None	
ERRORS	BadAccess	Lack of privilege
	BadWindow	Not a valid window
	BadValue	Not a valid input information label
SEE ALSO	XTSOLgetWindowIIL(3X11TSOL)	

NAME	XTSOLsetWorkstationOwner – Set the ownership of the workstation
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetWorkstationOwner (<i>display</i> , <i>uidp</i>) Display <i>*display</i> ; uid_t <i>*uidp</i> ;
AVAILABILITY	SUNWxwplt (available only on Trusted Solaris systems)
MT-LEVEL	Unsafe
DESCRIPTION	XTSOLsetWorkstationOwner() is used by Trusted Solaris logintool to assign a user ID to be identified as the owner of the workstation server. The client running under this user ID can set the server's device objects, such as keyboard mapping, mouse mapping, and modifier mapping. The client must have the Trusted Path process attribute.
ARGUMENTS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay . <i>uidp</i> Specifies a pointer to a <i>uid_t</i> structure that contains a UID.
RETURN VALUES	None
ERRORS	BadAccess Lack of privilege
SEE ALSO	XTSOLgetWorkstationOwner (3X11TSOL)

NAME	accept – Accept a connection on a socket								
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> int accept(int s, struct sockaddr *addr, int *addrlen);</pre>								
MT-LEVEL	Safe								
DESCRIPTION	The argument <i>s</i> is a socket that has been created with socket(3N) and bound to an address with bind(3N), and that is listening for connections after a call to listen(3N). accept() extracts the first connection on the queue of pending connections, creates a new socket with the properties of <i>s</i>, and allocates a new file descriptor, <i>ns</i>, for the socket. If no pending connections are present on the queue and the socket is not marked as nonblocking, accept blocks the caller until a connection is present. If the socket is marked as nonblocking and no pending connections are present on the queue, accept returns an error as described hereafter. accept uses the netconfig(4) file to determine the STREAMS device file name associated with <i>s</i>. This is the device on which the connect indication will be accepted. The accepted socket, <i>ns</i>, is used to read and write data to and from the socket that connected to <i>ns</i>; <i>ns</i> is not used to accept more connections. The original socket (<i>s</i>) remains open for accepting further connections. The argument <i>addr</i> is a result parameter that is filled with the address of the connecting entity as it is known to the communications layer. The exact format of the <i>addr</i> parameter is determined by the domain in which the communication occurs. <i>addrlen</i> is a value-result parameter. Initially, it contains the amount of space pointed to by <i>addr</i>; on return, <i>addrlen</i> contains the length in bytes of the address returned. accept is used with connection-based socket types, currently with SOCK_STREAM. It is possible to select(3C) or poll(2) a socket for the purpose of an accept by selecting or polling it for a read. However, this process will only indicate when a connect indication is pending; it is still necessary to call accept.								
RETURN VALUES	accept returns -1 on error. If it succeeds, accept returns a nonnegative integer that is a descriptor for the accepted socket.								
ERRORS	accept will fail if any of these conditions is true: <table> <tr> <td>EBADF</td><td>The descriptor is invalid.</td></tr> <tr> <td>EINTR</td><td>The accept attempt was interrupted by the delivery of a signal.</td></tr> <tr> <td>ENODEV</td><td>The protocol family and type corresponding to <i>s</i> could not be found in the netconfig file.</td></tr> <tr> <td>ENOMEM</td><td>There was insufficient user memory available to complete the operation.</td></tr> </table>	EBADF	The descriptor is invalid.	EINTR	The accept attempt was interrupted by the delivery of a signal.	ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the netconfig file.	ENOMEM	There was insufficient user memory available to complete the operation.
EBADF	The descriptor is invalid.								
EINTR	The accept attempt was interrupted by the delivery of a signal.								
ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the netconfig file.								
ENOMEM	There was insufficient user memory available to complete the operation.								

ENOSR	There were insufficient STREAMS resources available to complete the operation.
ENOTSOCK	The descriptor does not reference a socket.
EOPNOTSUPP	The referenced socket is not of type SOCK_STREAM .
EPROTO	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized or the connection has already been released.
EWouldBlock	The socket is marked as nonblocking and no connections are present to be accepted.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

If the calling process possesses the **PRIV_NET_MAC_READ** privilege and the socket has been bound to a multilevel port (MLP), the connection is accepted on a MLP; otherwise, the connection is accepted on a single-level port (SLP). [See **bind(3N)** for more information.]

SEE ALSO

poll(2), bind(3N), connect(3N), listen(3N), select(3C), socket(3N), netconfig(4)

NAME	adornfc – adorn the final component of a pathname														
SYNOPSIS	#include <tsol/mld.h> int adornfc(char *path_name, char *adorned_name)														
AVAILABILITY	SUNWtsolu														
MT-LEVEL	MT-Safe														
DESCRIPTION	adornfc() adorns the final component of <i>path_name</i> unless it is already adorned. <i>path_name</i> is a path name to a file system object. <i>adorned_name</i> is a pointer to a buffer in which the adorned version of <i>path_name</i> is placed. This buffer should be of at least MAX-PATHLEN bytes in length.														
RETURN VALUES	adornfc() returns: 0 on success. -1 on failure and sets errno to indicate the error.														
ERRORS	adornfc() fails if one or more of the following are true: <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path_name</i>.</td></tr> <tr> <td>EFAULT</td><td><i>path_name</i> or <i>adorned_name</i> points to an invalid address.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path_name</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the path argument exceeds {PATH_MAX} or MAX-PATHLEN. A pathname component is longer than {NAME_MAX} (see sysconf(2V)) while {_POSIX_NO_TRUNC} is in effect (see pathconf(2V)).</td></tr> <tr> <td>ENOENT</td><td>A component of the path prefix of <i>path_name</i> does not exist.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path prefix of <i>path_name</i> is not a directory.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> .	EFAULT	<i>path_name</i> or <i>adorned_name</i> points to an invalid address.	EIO	An I/O error occurred while reading from the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .	ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX} or MAX-PATHLEN . A pathname component is longer than {NAME_MAX} (see sysconf (2V)) while {_POSIX_NO_TRUNC} is in effect (see pathconf (2V)).	ENOENT	A component of the path prefix of <i>path_name</i> does not exist.	ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.
EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> .														
EFAULT	<i>path_name</i> or <i>adorned_name</i> points to an invalid address.														
EIO	An I/O error occurred while reading from the file system.														
ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .														
ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX} or MAX-PATHLEN . A pathname component is longer than {NAME_MAX} (see sysconf (2V)) while {_POSIX_NO_TRUNC} is in effect (see pathconf (2V)).														
ENOENT	A component of the path prefix of <i>path_name</i> does not exist.														
ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.														

NAME	au_class_x_t_alloc, au_class_x_t_free, au_class_x_t_cmp – Manipulate extended audit class
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> au_class_x_t *au_class_x_t_alloc(void) void *int au_class_x_t_free(void)</pre>
MT-LEVEL	Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	With no arguments (indicated by void), au_class_x_t_alloc() allocates an initialized, extended audit-mask variable for use by any of the extended audit-class library functions and system calls such as getaudit(2TSOL), au_preselect(3TSOL), getauusernam(3TSOL), and getfauditflags(3TSOL). au_class_x_t_free() deallocates the memory when the program is finished with it.
RETURN VALUES	Upon success, au_class_x_t_alloc returns a pointer to the allocated memory. Upon failure, au_class_x_t_alloc returns 0. au_class_x_t_free() does not return a value upon success or failure.
NOTES	This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for how to disable and enable auditing.
SEE ALSO	auditon(2TSOL) , getaudit(2TSOL) , au_preselect(3TSOL) , au_user_mask(3TSOL) , getauclassent(3TSOL) , getauditflags(3TSOL) , getauevent(3TSOL) , getauusernam(3TSOL) , getfauditflags(3TSOL)

NAME	au_preselect – Preselect an audit event										
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns -lintl [<i>library</i> ...]</pre> <pre>#include <bsm/libbsm.h></pre> <pre>int au_preselect(audit_event_t event, audit_mask_t *mask_p, int sorf, int flag);</pre>										
MT_LEVEL	MT-Safe										
AVAILABILITY	SUNWcsu										
DESCRIPTION	au_preselect determines whether or not the audit event <i>event</i> is preselected by the binary preselection mask pointed to by <i>mask_p</i> [usually obtained by a call to getaudit(2TSOL)]. au_preselect looks up the classes associated with <i>event</i> in audit_event(4TSOL) and compares them with the classes in <i>mask_p</i>. If the classes associated with <i>event</i> match the classes in the specified portions of the binary preselection mask pointed to by <i>mask_p</i>, the event is said to be preselected. <i>sorf</i> indicates whether the comparison is made with the success portion, the failure portion, or both portions of the mask pointed to by <i>mask_p</i>. These are the valid values of <i>sorf</i>: <table> <tr> <td>AU_PRS_SUCCESS</td><td>Compare the event class with the success portion of the preselection mask.</td></tr> <tr> <td>AU_PRS_FAILURE</td><td>Compare the event class with the failure portion of the preselection mask.</td></tr> <tr> <td>AU_PRS_BOTH</td><td>Compare the event class with both the success and the failure portions of the preselection mask.</td></tr> </table> <i>flag</i> tells au_preselect how to read the audit_event(4TSOL) database. Upon initial invocation, au_preselect reads the audit_event(4TSOL) database and allocates [with malloc(3C)] space in an internal cache for each entry. In subsequent invocations, the value of <i>flag</i> determines where au_preselect obtains audit event information. These are the valid values of <i>flag</i>: <table> <tr> <td>AU_PRS_REREAD</td><td>Get audit-event information by searching the audit_event(4TSOL) database.</td></tr> <tr> <td>AU_PRS_USECACHE</td><td>Get audit-event information from the internal cache created upon the initial invocation. This option is much faster.</td></tr> </table>	AU_PRS_SUCCESS	Compare the event class with the success portion of the preselection mask.	AU_PRS_FAILURE	Compare the event class with the failure portion of the preselection mask.	AU_PRS_BOTH	Compare the event class with both the success and the failure portions of the preselection mask.	AU_PRS_REREAD	Get audit-event information by searching the audit_event(4TSOL) database.	AU_PRS_USECACHE	Get audit-event information from the internal cache created upon the initial invocation. This option is much faster.
AU_PRS_SUCCESS	Compare the event class with the success portion of the preselection mask.										
AU_PRS_FAILURE	Compare the event class with the failure portion of the preselection mask.										
AU_PRS_BOTH	Compare the event class with both the success and the failure portions of the preselection mask.										
AU_PRS_REREAD	Get audit-event information by searching the audit_event(4TSOL) database.										
AU_PRS_USECACHE	Get audit-event information from the internal cache created upon the initial invocation. This option is much faster.										
RETURN VALUES	au_preselect returns one of these values: <table> <tr> <td>0</td><td><i>event</i> is not preselected.</td></tr> <tr> <td>1</td><td><i>event</i> is preselected.</td></tr> <tr> <td>-1</td><td>An error occurred. au_preselect could not allocate memory or could not find <i>event</i> in the audit_event(4TSOL) database.</td></tr> </table>	0	<i>event</i> is not preselected.	1	<i>event</i> is preselected.	-1	An error occurred. au_preselect could not allocate memory or could not find <i>event</i> in the audit_event(4TSOL) database.				
0	<i>event</i> is not preselected.										
1	<i>event</i> is preselected.										
-1	An error occurred. au_preselect could not allocate memory or could not find <i>event</i> in the audit_event(4TSOL) database.										

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

This function looks up the classes associated with *event* in **audit_event**(4TSOL) rather than in **audit_event**(4).

FILES

/etc/security/audit_class Maps audit-class number to audit-class names and descriptions
/etc/security/audit_event Maps audit-event number to audit-event names and associates

SEE ALSO

getaudit(2TSOL), **auditwrite**(3TSOL), **getauclassent**(3TSOL), **getauevent**(3TSOL), **malloc**(3C), **audit_class**(4TSOL), **audit_event**(4TSOL) *Trusted Solaris Audit Administration Manual*

NOTES

This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing. **au_preselect** is normally called prior to constructing and writing an audit record. If the event is not preselected, the overhead of constructing and writing the record can be saved.

NAME	au_user_mask – Get user's binary preselection mask
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> int au_user_mask(char *username, au_mask_t *mask_p);</pre>
MT-LEVEL	MT-Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	au_user_mask() reads the default, systemwide audit classes from audit_control(4TSOL), combines them with the per-user audit classes from the audit_user(4TSOL) database, and uses the combined value to update the binary preselection mask to which <i>mask_p</i> points. The audit flags in the <i>flags</i> field of the audit_control(4TSOL) database and the <i>always-audit-flags</i> and <i>never-audit-flags</i> from the audit_user(4TSOL) database represent binary audit classes. These fields are combined by au_preselect(3TSOL): $\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$ au_user_mask fails if the audit_control(4TSOL) entries cannot be retrieved. This mechanism allows for flexible configurations such that individual entries are not required in the audit_user(4TSOL) database.
RETURN VALUES	Upon success, au_user_mask returns 0. Upon failure, when both the audit_control (4TSOL) database entries cannot be retrieved, au_user_mask returns -1.
SUMMARY OF TRUSTED SOLARIS CHANGES	By default, the audit module is enabled on Trusted Solaris systems.
FILES	/etc/security/audit_control Contains default parameters read by the audit daemon, auditd(1MTSOL) /etc/security/audit_user Stores per-user audit-event mask
SEE ALSO	login (1), getaudit (2TSOL), setaudit (2TSOL), au_preselect (3TSOL), getacinfo (3TSOL), getauusernam (3TSOL), audit_control (4TSOL), audit_user (4TSOL)
NOTES	This functionality is active only if the audit module has been enabled. By default, this module is enabled on Trusted Solaris systems. Trusted Solaris 2.x will soon extend the number of audit classes and introduce new but similar structures and programming interfaces. au_user_mask should be called by programs like login(1TSOL) that set the preselection mask of a process with setaudit(2TSOL). getaudit(2TSOL) should be used to obtain audit characteristics for the current process.

NAME	auditwrite - Construct and write user-level audit records
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -lbsm -lsocket -lnsi -lintl -ltsol [<i>library</i>...] #include <bsm/auditwrite.h> #include <bsm/audit_uevents.h> #include <tsol/priv.h> #include <tsol/label.h> int auditwrite(..., AW_END);</pre>
AVAILABILITY	SUNWcsu
MT-LEVEL	MT-safe
DESCRIPTION	auditwrite() provides a single-function programmer interface to auditing for user-level programs. The principal features of auditwrite are audit record construction, audit record queueing, a save area, and support for trusted server auditing. See NOTES for privileges needed to write audit records.
Record Construction	auditwrite creates complete audit records or appends information to an existing, partial audit record. A single-shot audit record is one that is constructed and written in a single call to auditwrite. Multi-shot audit records are those constructed piecemeal through two or more calls to auditwrite. See EXAMPLES.
Record Queuing	Audit records may be queued by specifying a threshold. When the threshold is reached, records are written in one operation. This batching minimizes system-call overhead.
Save Area	A special audit-record buffer may be requested as a save area. Attributes stored in the save area are prepended to every subsequent record written with auditwrite.
Trusted Server Auditing Support	Some trusted servers act on behalf of untrusted client processes performing security checks and providing access to TCB objects. A trusted server must sometimes generate audit records with the audit characteristics of its clients. The AW_SERVER command tells auditwrite that the caller is a trusted server and that additional information should be added to each audit record if the information has not already been provided.
ARGUMENTS	auditwrite takes a variable number of arguments. Arguments are of three types. One type, referred to as control commands, controls the behavior of auditwrite. One and only one control command may appear on the auditwrite invocation line. Another type of argument is an attribute command. Attribute commands describe the attributes that compose an audit record. Control and attribute commands may appear in any order in an auditwrite invocation. The last type of argument is the terminator command. The terminator command AW_END must be the last argument on the auditwrite invocation line. The terminator command notifies auditwrite when to stop parsing the invocation line.

Control Commands**AW_WRITE**

Write to the audit trail the default audit record or the audit record specified by the last **AW_USERD** command. If queueing is in effect, the record is queued and written when the queue is flushed.

AW_APPEND

Attach one or more record attributes to the end of the record. **AW_APPEND** is used in the multi-shot construction of an audit record. Attributes are kept in a record buffer until **auditwrite** is called with the **AW_WRITE** command. One or more attribute commands must be specified with the **AW_APPEND** command.

AW_DEFAULTRD

Use the default record descriptor.

AW_DISCARD

Discard all partial and complete audit records.

AW_DISCARDRD, int *rd*

Discard the record descriptor specified by *rd*. An *rd* of **-1** can be specified to discard the default *rd*.

AW_GETRD, int **rd*

Obtain an audit record descriptor for use with **AW_USERD**.

AW_PRESELECT, au_mask_t **pmask*

Preselect audit records according to *pmask*. Normally audit records are preselected by **auditwrite** using the preselection mask in the execution environment. See **getaudit(2TSOL)** for details.

AW_NOPRESELECT

Use the preselection mask in the execution environment instead of the one specified with the **AW_PRESELECT** command.

AW_QUEUE, u_int *hi_water*

Turn on audit-record queueing. **AW_QUEUE** causes **auditwrite** to queue all audit records. When the total number of bytes of all queued audit records reaches *hi_water*, the queue is flushed. The queue may also be flushed at will with **AW_FLUSH**. The **auditwrite** audit-record queue should not be confused with the kernel audit-record queue. The **auditwrite** queue is a separate and distinct queue created in user address space. The **auditwrite()** audit-record queue can minimize system-call overhead by writing several audit records in one operation.

AW_NOQUEUE

Turn off audit-record queueing. Flush the queue.

AW_FLUSH

Flush the audit-record queue.

AW_SAVERD, int **rd*

Turn on use of a save area. Attributes stored in the save area are prepended to records before they are written.

AW_NOSAVE

Turn off use of the save area.

AW_SERVER

Turn on the trusted server option. The **AW_SERVER** command tells **auditwrite** that the calling program is a server that must generate complete audit records. **auditwrite** adds header and trailer attributes to all records. Sequence and group attributes are also added depending upon the audit policy. See **audit(2TSOL)** for further information on audit policy.

AW_NOSERVER

Turn off the trusted server option.

AW_USERD int rd

Use the record descriptor obtained with **AW_GETRD**.

Attribute Commands

Attribute commands describe the attributes that compose an audit record. Attribute commands must be specified with one and only one **AW_APPEND** or **AW_WRITE** control command.

AW_ARG, char n, char *text, u_long v

Place the specified system call argument information into the audit record. *n* contains the argument number. *text* contains a string describing the argument. *v* contains the value of the argument.

AW_ATTR, mode_t mode, uid_t uid, gid_t gid, dev_t dev, ino_t ino, dev_t rdev

Place the specified file system object attribute information into the audit record. You can get this information using the **stat(2)** system call.

AW_CLEARANCE, bclear_t *clear

Place the specified clearance into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from **auditon(2TSOL)** is not enabled on this system, the command is ignored.

AW_DATA, char unit_print, char unit_type, char unit_count, caddr_t p

Place the specified arbitrary data into the audit record. *unit_print* describes how the data should be printed by programs that read the audit trail. These are allowable values for *unit_print*:

AWD_BINARY

AWD_OCTAL

AWD_DECIMAL

AWD_HEX

AWD_STRING

unit_type describes the type of data in *p*. These are allowable values for *unit_type*:

AWD_BYTE

AWD_CHAR

AWD_SHORT

AWD_INT

AWD_LONG

unit_count describes how many elements of *unit_type* exist in *p*, which is an address pointing to the data to be written.

AW_EVENT, char *event_str

Specify the audit event associated with the audit record. One and only one event must be associated with every audit record. If an attempt is made to write an audit record for which an event has not been specified, **auditwrite** returns an error. *event_str* is any valid user-level audit-event string as defined by **audit_event**(4TSOL).

(**AW_EVENT** is used for third-party application events. For other events, **AW_EVENTNUM** should be used if possible, since **AW_EVENT** incurs additional overhead for string lookup.)

AW_EVENTNUM, int event

AW_EVENTNUM is similar to **AW_EVENT** but takes as its argument any valid *event* number instead of an event string. To maintain compatibility between third party add-ons, only registered events may use this attribute command. *event* is any valid audit event defined in **audit_event**(4TSOL) and **/usr/include/bsm/audit_uevent.h**. See **audit_event**(4TSOL) for further information on audit-event strings.

AW_EXEC_ARGS, char **argv

Place the specified command line arguments into the audit record. The array is terminated by a null pointer. (The format is the same as that used for *argv* by an invoking C program.) If the appropriate audit policy (**argv**) from **auditon**(2TSOL) is not enabled on this system, this call is ignored.

AW_EXEC_ENV, char **envp

Place the specified command-line environment into the audit record. The array is terminated by a null pointer. (This format is the same as that used for *envp* by an invoking C program.) If the appropriate audit policy (**argv**) from **auditon**(2TSOL) is not enabled on this system, this call is ignored.

AW_EXIT, int status, int errno

Place the specified program exit-status information into the audit record. *status* contains the exit status of the calling program. *errno* contains the system error number or an internal error number indicating the cause of the program exit.

AW_GROUPS, int num, gid_t *groups

Format and place the elements of the array *groups* into the audit record. *num* specifies the number of elements in the array and must be between **NGROUPS_UMIN** and **NGROUPS_UMAX** as defined in **/usr/include/sys/param.h**. If the audit policy [see **auditconfig**(1MTSOL)] is not configured for including supplementary groups, the command is ignored.

AW_ILABEL, bilabel_t *label

Place the specified information label into the audit record. If information labels are not enabled on this system via **system**(4) or if the appropriate audit policy (*ilabel*) from **auditon**(2TSOL) is not enabled on this system, this call is also ignored.

AW_INADDR, struct in_addr *in_addr

Place the specified Internet address into the audit record.

AW_IPC, char type, int id

Place the specified interprocess-communications identifier into the audit record. *type* is one of these values: **AT_IPC_MSG**, **AT_IPC_SEM**, **AT_IPC_SHM**, or **AT_IPC_NULL**.

AW_IPC_PERM, **struct ipc_perm** **perm*

Place the specified interprocess-communications identifier permission information into the audit record.

AW_IPORT, **u_short** *iport*

Place the specified IP port into the audit record.

AW_LEVEL, **blevel_t** **level*

Place the specified level into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from **auditon**(2TSOL) is not enabled on this system, the command is ignored.

AW_OPAQUE, **caddr_t** *data*, **short** *byte_count*

Place into the audit record the opaque data to which *data* points and which has *byte_count* length.

AW_PATH, **char** **path*

Place the specified path into the audit record. Paths are anchored with the current active root if they do not begin with a slash (/).

AW_PRIVILEGE, **priv_set_t** **priv_set*, **char** *settype*

Place the specified privilege set into the audit record. These are allowable values for *settype*:

AU_PRIV_UNKNOWN

AU_PRIV_FORCED

AU_PRIV_ALLOWED

AU_PRIV_EFFECTIVE

AU_PRIV_INHERITABLE

AU_PRIV_PERMITTED

AU_PRIV_SAVED

AW_PROCESS, **au_id_t** *auid*, **uid_t** *euid*, **gid_t** *egid*, **uid_t** *ruid*, **gid_t** *rgid*, **pid_t** *pid*,
au_asid_t *sid*, **au_tid_t** **tid*

Place the specified process information into the audit record. The **AW_PROCESS** and **AW_SUBJECT** attributes record the same information. Use the **AW_PROCESS** attribute when recording information about a process object. Use the **AW_SUBJECT** attribute when recording information about a process subject.

AW_RETURN, **char** *number*, **u_int** *retval*

The **AW_RETURN** attribute indicates the success or failure of an audit event. This attribute is used by **auditwrite**() for preselection, and by the **auditreduce**(1MTSOL) post-selection program to select audit records according to success or failure.

number indicates the success or failure of the event. Failure is denoted by a nonzero *number* value. Positive values are interpreted by **praudit**(1MTSOL) as **errno** values. Corresponding error strings are printed. Negative values indicate a general failure specific to the audit event. Success is denoted by a zero *number*

value. *retval* indicates the return value or status value of the successful or failed function or program.

AW_SLABEL, bslabel_t *label

Place the specified sensitivity label into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from **auditon**(2TSOL) is not enabled on this system, the command is ignored.

AW_SOCKET, struct socket *s

Place the specified socket information into the audit record.

AW_SUBJECT, au_id_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid, pid_t pid, au_aside_t sid, au_tid_t *tid

Place the specified subject information into the audit record. The **AW_PROCESS** and **AW_SUBJECT** attributes record the same information. Use the **AW_PROCESS** attribute when recording information about a process object. Use the **AW_SUBJECT** attribute when recording information about a process subject.

AW_TEXT, char *text

Place the specified null-terminated string *text* into the audit record.

AW_USEOFFPRIV, char flag, priv_t priv

Place a flag and a single privilege into the audit record denoting an attempted use of privilege. *flag* indicates success (**1**) or failure (**0**) of the attempt. *priv* indicates the privilege upon which the attempt was made.

AW_XATOM, char *atom_string

Place the specified X atom string into the audit record.

AW_XCOLORMAP, u_long xid, uid_t creator_uid

Place the specified X colormap information into the audit record.

AW_XCLIENT, u_long clientid

Place the specified X client ID into the audit record.

AW_XCURSOR, u_long xid, uid_t creator_uid

Place the specified X cursor information into the audit record.

AW_XFONT, u_long xid, uid_t creator_uid

Place the specified X font information into the audit record.

AW_XGC, u_long xid, uid_t creator_uid

Place the specified X gc information into the audit record.

AW_XPIXMAP, u_long xid, uid_t creator_uid

Place the specified X pixel-mapping information into the audit record.

AW_XPROPERTY, u_long xid, uid_t creator_uid, char *atom_name

Place the specified X property information into the audit record.

AW_XSELECT, char *property_string, char *property_type, char *window_data

Place the specified X select information into the audit record.

AW_XWINDOW, u_long xid, uid_t creator_uid

Place the specified X window information into the audit record.

**Terminator
Command**

The terminator command **AW_END** must be the last argument on the **auditwrite** invocation line.

RETURN VALUES

Upon success, **auditwrite** returns **0**. Upon failure, **auditwrite** returns a value of **-1** and sets **aw_errno** to indicate the error.

ERRORS

When an error is encountered, any whole or partial audit records are immediately written to the audit trail. These include records that may have been queued. In addition, a **LOG_ALERT** message is sent to **syslogd(1M)** and an attempt is made to write an **auditwrite** “processing error” audit record to the audit trail.

aw_strerror(3TSOL) or **aw_perror(3TSOL)** may be used for obtaining the error strings associated with **aw_errno**.

These are the possible values of **aw_errno**:

AW_ERR_ADDR_INVALID	An address specified was invalid.
AW_ERR_ALLOC_FAIL	An attempt to allocate memory failed.
AW_ERR_AUDITON_FAIL	The auditon(2TSOL) system call failed. See errno for the reason.
AW_ERR_AUDIT_FAIL	The audit(2TSOL) system call failed. See errno for the reason.
AW_ERR_CMD_INCOMPLETE	A required command was omitted. This event occurs when AW_APPEND is specified without any attribute commands or vice versa.
AW_ERR_CMD_INVALID	The command specified was not a valid command.
AW_ERR_CMD_IN_EFFECT	A command already in effect, such as AW_QUEUE or AW_SAVERD was specified.
AW_ERR_CMD_NOT_IN_EFFECT	An attempt was made to reverse a command that was not in effect.
AW_ERR_CMD_TOO_MANY	More than one control command was specified.
AW_ERR_EVENT_ID_INVALID	The event ID string passed was not valid.
AW_ERR_EVENT_ID_NOT_SET	An attempt was made to write an audit record without a valid event ID. When this attempt occurs, the event ID is set to a null value and the record is written anyway as a part of error-processing procedure.
AW_ERR_GETAUDIT_FAIL	The getaudit(2TSOL) system call failed. See errno for the reason.
AW_ERR_QUEUE_SIZE_INVALID	The specified queue size was greater than the system-imposed maximum audit-record size.
AW_ERR_RD_INVALID	The specified record descriptor was invalid.
AW_ERR_REC_TOO_BIG	An attempt was made to construct an audit record

larger than the system-imposed maximum audit-record size.

AW_ERR_NO_PLABEL

The process label for the current process could not be obtained; thus a complete record could not be generated.

EXAMPLES **Valid Examples**

/* Single-shot record construction:

*** Construct an audit record and write the record to the audit trail.**

*** Uses the default audit record.**

***/**

```
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string1, AW_TEXT, "hello", AW_WRITE, AW_END);
```

/* Multi-shot construction:

*** Construct an audit record piecemeal and write the record.**

*** Uses the default audit record.**

***/**

```
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string2, AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_TEXT, "part 1", AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_TEXT, "part 2", AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_RETURN, 0, 0, AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_WRITE, AW_END);
```

/* Multi-shot record construction:

*** Decide upon the return token value when it occurs.**

***/**

```
(void) auditwrite(AW_EVENTNUM, AUE_ftpd, AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_TEXT, "Read access attempt", AW_APPEND, AW_END);
```

```
if (access_decision() == FALSE) {
```

```
    succ_or_fail = -1;
```

```
    reason = get_reason;
```

```
} else {
```

```
    succ_or_fail = 0;
```

```
    reason = 0;
```

```
}
```

```
(void) auditwrite(AW_TEXT, "more text", AW_RETURN, succ_or_fail, reason, AW_APPEND, AW_END);
```

```
(void) auditwrite(AW_WRITE, AW_END);
```

/* Queueing:

Invalid Examples

```

* Turn on queueing, queue two records, then turn off queueing. Queue is flushed
* automatically when queueing is turned off.
*/

(void) auditwrite(AW_QUEUE, 1024, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string3, AW_RETURN, 0, 0, AW_WRITE, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string4, AW_RETURN, 0, 0, AW_WRITE, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string5, AW_RETURN, 0, 0, AW_APPEND, AW_END);
(void) auditwrite(AW_NOQUEUE, AW_END);

/*
* Invalid command combinations:
* Only one control command may be specified.
*/

(void) auditwrite(AW_EVENTNUM, valid_event_str, AW_TEXT, "text", AW_DISCARD, AW_WRITE, AW_END);

/*
* No control command specified
*/
(void) auditwrite(AW_TEXT, "text", AW_END);

```

FILES

audit_event(4TSOL) Used to obtain the mappings between audit-event strings and audit-event numbers

SEE ALSO

auditreduce(1MTSOL), **praudit**(1MTSOL), **audit**(2TSOL), **getaudit**(2TSOL), **stat**(2TSOL), **audit_event**(4TSOL), **auditconfig**(1MTSOL), **auditon**(2TSOL) *Security Features Guide*

NOTES

When a subject is not provided, **auditwrite** attempts to generate the subject, groups, and (depending on appropriate audit policy) sensitivity label, and information label attributes for the current process. This attempt has implications for servers because unless they negotiate to get the AUID, UID, sensitivity label, information label, and groups of the process being served and provide them to **auditwrite**, the values recorded will be those of the server process. Programmers of servers should take this circumstance into account when using **auditwrite** and make specific requests to **auditwrite** to work around this problem.

To write an audit record with an event number from 2048 to 32767, the calling process must have **PRIV_PROC_AUDIT_TCB** in its set of effective privileges. If the event number is from 32768 to 65535, the calling process must have **PRIV_PROC_AUDIT_APPL** in its set of effective privileges. These sets of event numbers are the only valid user-level event numbers.

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – translate and verify user authorizations
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol -ltsolddb -lcmd -lnsl [library ...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id) char *auth_set_to_str(auth_set_t *authset, char separator) auth_t str_to_auth(char *auth_name) auth_set_t *str_to_auth_set(char *auth_names, char *separators) char *get_auth_text(auth_t auth_id) int chkauth(auth_t auth_id, char *username) void free_auth_set(auth_set_t *authset)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i>. If <i>auth_id</i> is an undefined authorization ID, the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID, TSOL_MAX_AUTH, a NULL is returned. auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C). Authorization names are separated by the character <i>separator</i>. Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list. str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i>. <i>auth_name</i> is the name in the Names field of auth_name(4TSOL); this function does not parse the names in the Manifest Constant field of auth_name(4TSOL). Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH. str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i>. <i>auth_names</i> are names in the Names field of auth_name(4TSOL); this function does not parse the names in the Manifest Constant field of auth_name(4TSOL). The string none is translated to an empty authorization list (NULL). This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization names appear in the <i>auth_names</i> string, the function converts these invalid authorizations into <i>auth_ids</i> with the value of 0.

	get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by <i>auth_id</i>. chkauth returns 1 if the user, specified by <i>username</i> has the authorization specified by <i>auth_id</i>. free_auth_set releases the memory returned by str_to_auth_set().
RETURN VALUES	auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure. str_to_auth() returns the numeric authorization id. It returns 0 on failure. auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure. str_to_auth_set() returns NULL on success. get_auth_text() return a string on success and NULL upon failure. chkauth () returns 1 on success and 0 on failure.
NOTES	This interface is uncommitted, which means it may change between minor releases of Trusted Solaris 2.x. To use these routines, the program must be loaded with the Trusted Solaris library libtsolddb.
SEE ALSO	auth_desc(4TSOL) auth_name(4TSOL)

NAME	aw_errno, aw_strerror, aw_perror – Obtain and display auditwrite (3TSOL) error messages
SYNOPSIS	<pre>cc [flag ...] file... -lbsm -lsocket -lns -lintl -ltsol [library...] #include <bsm/auditwrite.h> int aw_errno; char *aw_strerror(const int status); void aw_perror(const char *s);</pre>
MT-LEVEL	MT-Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	<i>aw_errno</i> is a variable set by auditwrite(3TSOL) to indicate the type of error encountered during its execution, in much the same manner as errno(3C) is set during a system or library call. The supporting functions convert auditwrite(3TSOL) error numbers into text strings. aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string. aw_perror() prints the error message corresponding to <i>status</i> as “<i>s: error_message</i>” on standard error.
SEE ALSO	errno (3C), perror (3C), strerror (3C), auditwrite (3TSOL)
NOTES	The returned string must not be overwritten. If string manipulation is required, work on a local copy.

NAME	bcltobanner – translate binary CMW labels to character-coded labels for a printer banner page				
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> struct banner_fields { char *header; /* top and bottom banner/trailer page */ char *protect_as; /* “protect as” banner page section */ char *ilabel; /* Information Label, also top and bottom of body pages */ char *caveats; /* “caveats” banner page section */ char *channels; /* “handling channels” section */ /* lengths of pre-allocated string memory */ short header_len; /* header */ short protect_as_len; /* protect_as */ short ilabel_len; /* Information Label */ short caveats_len; /* caveats */ short channels_len; /* handling channels */ }; int bcltobanner(const bclabel_t *label, struct banner_fields</pre>				
AVAILABILITY	SUNWtsolu				
MT-LEVEL	MT-Safe				
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on labels that dominate the current processes' sensitivity label. bcltobanner() translates a <i>Binary CMW Label</i>, label, into various character-coded labels and strings for display on printer banner and trailer pages and at the top and bottom of the document body pages. The members of the <i>fields</i> structure are either string pointers, or the length of memory pre-allocated to a string pointer. The string pointers may contain either a pointer to pre-allocated memory, or the value (char *)0. If the string pointer contains a pointer to pre-allocated memory, then its associated length member indicates the size of that memory. If it contains the value (char *)0, memory is allocated using malloc() to contain the translated character-coded label or string. The translated string is copied into allocated or pre-allocated memory. Members of the fields structure have the following meaning: <table> <tr> <td><i>header</i></td><td>String to print at the top and bottom of banner and trailer pages</td></tr> <tr> <td><i>protect_as</i></td><td>String to print in the <i>protect as</i> warning of banner and trailer pages</td></tr> </table>	<i>header</i>	String to print at the top and bottom of banner and trailer pages	<i>protect_as</i>	String to print in the <i>protect as</i> warning of banner and trailer pages
<i>header</i>	String to print at the top and bottom of banner and trailer pages				
<i>protect_as</i>	String to print in the <i>protect as</i> warning of banner and trailer pages				

ilabel String to print in the *this system has labeled* message of banner and trailer pages, and at the top and bottom of the document body pages

caveats String to print in *caveats* section of the banner and trailer pages

channels String to print in *channels* section of the banner and trailer pages

header_len Length of pre-allocated memory for *header* string

protect_as_len Length of pre-allocated memory for *protect_as* string

ilabel_len Length of pre-allocated memory for *ilabel* string

caveats_len Length of pre-allocated memory for *caveats* string

channels_len Length of pre-allocated memory for *channels* string

The translation is controlled by the value of the **flags** parameter. **flags** may be either **LONG_WORDS**, **SHORT_WORDS**, or 0 (zero).

LONG_WORDS Translate using long names of *Words* defined in **label**.

SHORT_WORDS

Translate using short names of *Words* defined in **label**. If no short name is defined in the **label_encodings** file for a *Word*, the long name is used.

A **flags** value 0 is equivalent to **LONG_WORDS**.

RETURN VALUES

bcltobanner returns:

- 1** If **label** is not a *Binary CMW Label* with a defined information label and sensitivity label, or if its information label portion is not dominated by its sensitivity label portion, or if its sensitivity label portion is not dominated by the process sensitivity label and the process does not have **PRIV_SYS_TRANS_LABEL** in its set of effective privileges, or if the **label_encodings** file is inaccessible.
- 0** If memory cannot be allocated for a string in the *fields* structure, or if one of the pre-allocated memories is insufficient to hold its string. The value of that pre-allocated string is set to the **NULL string** (*fields->string[0]* = **'\000'**).
- 1** If successful.

EXAMPLES

The string members of the *fields* structure are included in a printer banner page in the following manner:

HEADER

This output must be protected as:

PROTECT_AS

unless manually reviewed and downgraded.

The system has labeled this data:

ILABEL

CAVEATS
CHANNELS

HEADER

The *ilabel* member is also placed at the top and bottom of the body page.

FILES **/etc/security/tsol/label_encodings**

The label encodings file containing the *Classification* names, *Words*, constraints, and values for the defined labels of this system.

SEE ALSO **bilconjoin(3TSOL)C , blcompare(3TSOL)C , blinset(3TSOL)C , blmanifest(3TSOL)C , blminmax(3TSOL)C , blportion(3TSOL)C , bltos(3TSOL)C , bltype(3TSOL)C , blvalid(3TSOL)C , btohex(3TSOL)C ,**

NAME	bilconjoin – conjoin binary information labels
SYNOPSIS	cc [<i>flag ...</i>] <i>file ...</i> -ltsol [<i>library ...</i>] #include <tsol/label.h> void bilconjoin(bilabel_t *receiving_label, const bilabel_t *adding_label)
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	bilconjoin() replaces the contents of binary information label <i>receiving_label</i> with the conjunction of binary information labels <i>receiving_label</i> and <i>adding_label</i> .
SEE ALSO	bcltobanner(3TSOL)C , blcompare(3TSOL)C , blinset(3TSOL)C , blmanifest(3TSOL)C , blminmax(3TSOL)C , blportion(3TSOL)C , bltcolor(3TSOL)C , bltos(3TSOL)C , bltype(3TSOL)C , blvalid(3TSOL)C , btohex(3TSOL)C , hextob(3TSOL)C , , labelinfo(3TSOL)C , labelvers(3TSOL)C , sbltos(3TSOL)C , stobl(3TSOL) <i>Trusted Solaris Developer's Guide</i>

NAME	bind – Bind a name to a socket																										
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> int bind(int s, const struct sockaddr *name, int namelen);</pre>																										
MT-LEVEL	Safe																										
DESCRIPTION	bind() assigns a name to an unnamed socket. When it is created with socket(3N) , a socket exists in a name space (address family) but has no name assigned. bind requests that the name to which <i>name</i> points be assigned to the socket.																										
RETURN VALUES	If the bind is successful, 0 is returned. A return value of -1 indicates an error, which is further specified in the global errno .																										
ERRORS	The bind call will fail if any of these conditions is true: <table> <tr> <td>EACCES</td><td>The requested address is protected and the current user has inadequate permission to access it.</td></tr> <tr> <td>EADDRINUSE</td><td>The specified address is already in use.</td></tr> <tr> <td>EADDRNOTAVAIL</td><td>The specified address is not available on the local machine.</td></tr> <tr> <td>EBADF</td><td><i>s</i> is not a valid descriptor.</td></tr> <tr> <td>EINVAL</td><td><i>namelen</i> is not the size of a valid address for the specified address family.</td></tr> <tr> <td>EINVAL</td><td>The socket is already bound to an address.</td></tr> <tr> <td>ENOSR</td><td>There were insufficient STREAMS resources for the operation to complete.</td></tr> <tr> <td>ENOTSOCK</td><td><i>s</i> is a descriptor for a file, not a socket.</td></tr> </table> The following errors are specific to binding names in the UNIX domain: <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of the path name in <i>name</i>.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while making the directory entry or allocating the inode.</td></tr> <tr> <td>EISDIR</td><td>A null path name was specified.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating the path name in <i>name</i>.</td></tr> <tr> <td>ENOENT</td><td>A component of the path prefix of the path name in <i>name</i> does not exist.</td></tr> </table>	EACCES	The requested address is protected and the current user has inadequate permission to access it.	EADDRINUSE	The specified address is already in use.	EADDRNOTAVAIL	The specified address is not available on the local machine.	EBADF	<i>s</i> is not a valid descriptor.	EINVAL	<i>namelen</i> is not the size of a valid address for the specified address family.	EINVAL	The socket is already bound to an address.	ENOSR	There were insufficient STREAMS resources for the operation to complete.	ENOTSOCK	<i>s</i> is a descriptor for a file, not a socket.	EACCES	Search permission is denied for a component of the path prefix of the path name in <i>name</i> .	EIO	An I/O error occurred while making the directory entry or allocating the inode.	EISDIR	A null path name was specified.	ELOOP	Too many symbolic links were encountered in translating the path name in <i>name</i> .	ENOENT	A component of the path prefix of the path name in <i>name</i> does not exist.
EACCES	The requested address is protected and the current user has inadequate permission to access it.																										
EADDRINUSE	The specified address is already in use.																										
EADDRNOTAVAIL	The specified address is not available on the local machine.																										
EBADF	<i>s</i> is not a valid descriptor.																										
EINVAL	<i>namelen</i> is not the size of a valid address for the specified address family.																										
EINVAL	The socket is already bound to an address.																										
ENOSR	There were insufficient STREAMS resources for the operation to complete.																										
ENOTSOCK	<i>s</i> is a descriptor for a file, not a socket.																										
EACCES	Search permission is denied for a component of the path prefix of the path name in <i>name</i> .																										
EIO	An I/O error occurred while making the directory entry or allocating the inode.																										
EISDIR	A null path name was specified.																										
ELOOP	Too many symbolic links were encountered in translating the path name in <i>name</i> .																										
ENOENT	A component of the path prefix of the path name in <i>name</i> does not exist.																										

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES****SEE ALSO****NOTES****ENOTDIR**

A component of the path prefix of the path name in *name* is not a directory.

EROFS

The inode would reside on a read-only file system.

If the calling process possesses the **PRIV_NET_MAC_READ** privilege, the socket is bound to a multilevel port (MLP); otherwise, the socket is bound to a single-level port (SLP).

unlink(2TSOL), **socket(3N)**

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller [using **unlink(2)** when it is no longer needed.

The rules used in name binding vary among communication domains.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – compare binary labels
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ltsol [<i>library</i> ...] #include <tsol/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t *label1, const blevel_t *label2) int bilequal(const bilabel_t *label1, const bilabel_t *label2) int bimequal(const bilabel_t *label1, const bilabel_t *label2) int bldominates(const blevel_t *label1, const blevel_t *label2) int blstrictdom(const blevel_t *label1, const blevel_t *label2) int bildominates(const bilabel_t *label1, const bilabel_t *label2) int bimdominates(const bilabel_t *label1, const bilabel_t *label2) int blinrange(const blevel_t *label, const brange_t *range)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions compare binary labels for meeting a particular condition. blequal() compares two levels for equality. bilequal() compares two information labels for equality. bimequal() compares two information label markings sets for equality. bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>. blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>. bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i>. bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i>. blinrange() compares level <i>label</i> for dominance over <i>range</i>—><i>lower_bound</i> and <i>range</i>—><i>upper_bound</i> for dominance over level <i>label</i>.
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.
EXAMPLES	The following example shows how to compare all parts of two binary labels.

blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) &&
bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))

SEE ALSO

bcltobanner(3TSOL)C , **bilconjoin**(3TSOL)C , **blinset**(3TSOL)C , **blmanifest**(3TSOL)C ,
blminmax(3TSOL)C , **blportion**(3TSOL)C , , **bltos**(3TSOL)C , **bltype**(3TSOL)C ,
blvalid(3TSOL)C , **btohex**(3TSOL)C , **hextob**(3TSOL)C , **labelinfo**(3TSOL)C ,
labelvers(3TSOL)C , **sbltos**(3TSOL)C , **stobl**(3TSOL)

Trusted Solaris Developer's Guide

NAME	blinset – check binary label for inclusion in set
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> typedef label_set_identifier { int type; char *name; } set_id; int blinset(const bslabel_t *label, const set_id *id)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform tests on labels that dominate the current processes' sensitivity label. <i>label</i> is examined to determine if it is an element of the label set <i>id</i>. The <i>set_id type</i> field contains a manifest constant defining the type of set to be examined. The <i>set_id name</i> field contains the name of the particular set of type <i>type</i>. The following types and names are defined: SYSTEM_ACCREDITATION_RANGE the system's accreditation range as defined in the label_encodings file. The <i>name</i> field is ignored and need not be specified. USER_ACCREDITATION_RANGE the user accreditation range as defined in the label_encodings file. The <i>name</i> field is ignored and need not be specified. Other <i>type</i> and <i>name</i> values are reserved for future implementation.
RETURN VALUES	blinset() returns: 1 If the label is contained in the specified set. 0 If the label is not a valid sensitivity label, is not dominated by the process sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges, or is not in the specified set. -1 If the specified set is inaccessible.
FILES	/etc/security/tsol/label_encodings The label encodings file containing the <i>Classification</i> names, <i>Words</i>, constraints, and values for the defined labels of this system.

SEE ALSO	bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL), label_encodings(4TSOL) <i>Trusted Solaris Developer's Guide</i> <i>Trusted Solaris administrator's document set</i>
NOTES	The <i>Admin Low</i> and <i>Admin High</i> labels are accepted even if the remainder of the System's Accreditation Range is inaccessible.
BUGS	The only sets available are the System's and User Accreditation Range.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – create manifest Binary Labels
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> void bcllow(bclabel_t *label) void bclhigh(bclabel_t *label) void bsllow(bsllabel_t *label) void bsllhigh(bsllabel_t *label) void billow(bilabel_t *label) void bilhigh(bilabel_t *label) void bclearlow(bclear_t *clearance) void bclearhigh(bclear_t *clearance) void bclundef(bclabel_t *label) void bsllundef(bsllabel_t *label) void bilundef(bilabel_t *label) void bclearundef(bclear_t *clearance)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions initialize <i>Binary Labels</i> structures to manifest values. bcllow() and bclhigh() initialize the <i>Binary CMW Label</i> structure <i>label</i> to the manifest constant values for the <i>Admin Low</i> and <i>Admin High CMW Labels</i> respectively. bsllow() and bsllhigh() initialize the <i>Binary Sensitivity Label</i> structure <i>label</i> to the manifest constant values for the <i>Admin Low</i> and <i>Admin High Sensitivity Labels</i> respectively. billow() and bilhigh() initialize the <i>Binary Information Label</i> structure <i>label</i> to the manifest constant values for the <i>Admin Low</i> and <i>Admin High Information Labels</i> respectively. bclearlow() and bclearhigh() initialize the <i>Binary Clearance</i> structure <i>clearance</i> to the manifest constant values for the <i>Admin Low</i> and <i>Admin High Clearances</i> respectively. bclundef(), bsllundef(), and bilundef() initialize the <i>Binary CMW</i>, <i>Sensitivity</i>, and <i>Information Label</i> structure <i>label</i> to the manifest constant value for an <i>Undefined CMW</i>, <i>Sensitivity</i>, and <i>Information Label</i> respectively. bclearundef() initializes the <i>Binary Clearance</i> <i>clearance</i> to the manifest constant value for an <i>Undefined Clearance</i>.

SEE ALSO

bcltobanner(3TSOL), **bilconjoin**(3TSOL), **blcompare**(3TSOL), **blinset**(3TSOL), **blminmax**(3TSOL), **blportion**(3TSOL), **bltos**(3TSOL), **bltype**(3TSOL), **blvalid**(3TSOL), **btohex**(3TSOL), **hextob**(3TSOL), **labelinfo**(3TSOL), **labelvers**(3TSOL), **sbltos**(3TSOL), **stobl**(3TSOL)

Trusted Solaris Developer's Guide

NAME	blminmax, blmaximum, blminimum – bound of two binary levels
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> void blmaximum(blevel_t *maximum_label, const blevel_t *bounding_label) void blminimum(blevel_t *minimum_label, const blevel_t *bounding_label)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	blmaximum() replaces the contents of <i>Binary Level maximum_label</i> with the least upper bound of <i>Binary Levels maximum_label</i> and <i>bounding_label</i>. The least upper bound is the greater of the <i>Classifications</i> and all of the <i>Compartments</i> of the two <i>Binary Levels</i>. This is the least <i>Binary Level</i> that dominates both the original <i>Binary Levels</i>. blminimum() replaces the contents of <i>Binary Level minimum_label</i> with the greatest lower bound of <i>Binary Levels minimum_label</i> and <i>bounding_label</i>. The greatest lower bound is the lower of the <i>Classifications</i> and only the <i>Compartments</i> contained in both <i>Binary Levels</i>. This is the greatest <i>Binary Level</i> that is dominated by both the original <i>Binary Levels</i>.
SEE ALSO	bcltobanner(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blportion(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), hextob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL) <i>Trusted Solaris Developer's Guide</i>

NAME	blportion, bcltosl, bcltoil, biltolev, getcsl, getcil, setcsl, setcil – access binary label portions
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> bslabel_t *bcltosl(bclabel_t *label) bilabel_t *bcltoil(bclabel_t *label) blevel_t *biltolev(bilabel_t *label) void getcsl(bslabel_t *destination_label, const bclabel_t *source_label) void getcil(bilabel_t *destination_label, const bclabel_t *source_label) void setcsl(bclabel_t *destination_label, const bslabel_t *source_label) void setcil(bclabel_t *destination_label, const bilabel_t *source_label)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions provide pointers to, extract, and replace portions of <i>Binary Labels</i>. bcltosl() and bcltoil() provide a pointer to the <i>Sensitivity Label</i> and <i>Information Label</i> portion of the <i>Binary CMW Label</i> <i>label</i> respectively. biltolev() provides a pointer to the <i>Information Level</i> portion of the <i>Binary Information Label</i> <i>label</i>. getcsl() and getcil() copies the <i>Sensitivity Label</i> and <i>Information Label</i> portion of the <i>Binary CMW Label</i> <i>source_label</i> to the <i>Binary Sensitivity Label</i> and <i>Binary Information Label</i> <i>destination_label</i> respectively. setcsl() and setcil() replaces the value of the <i>Sensitivity Label</i> and <i>Information Label</i> portion of the <i>Binary CMW Label</i> <i>destination_label</i> with the value of the <i>Binary Sensitivity Label</i> and <i>Binary Information Label</i> <i>source_label</i> respectively.
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types. biltolev() returns a pointer to the level portion of the parameter <i>Information Label</i>.
EXAMPLES	
bcltosl	The following example shows how to compare the <i>Sensitivity Label</i> portion of a <i>Binary CMW Label</i> with a file's <i>Binary Sensitivity Label</i>. <pre>blequal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>
bcltoil	The following example shows how to conjoin a file's <i>Information Label</i> with the <i>Information Label</i> portion of a <i>Binary CMW Label</i>.

bilconjoin(bcltoil(&cmw_label), &file_information_label)**biltolev**

The following example shows how to see if the *Information Label* portion of a *Binary CMW Label* is dominated by the *Sensitivity Label* portion.

bldominates(bcltosl(&cmw_label), biltolev(bcltoil(&cmw_label)))**SEE ALSO**

bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), bltcolor(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), hex tob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sb ltos(3TSOL), stobl(3TSOL)

Trusted Solaris Developer's Guide

NAME	bltcolor, bltcolor_r – get ASCII color name of label
SYNOPSIS	<pre>#include <tsol/label.h> char *bltcolor(const blevel_t *label) char *bltcolor_r(const blevel_t *label, const int size, char *color_name)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe with exceptions described in NOTES .
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to get color names of labels that dominate the current processes' sensitivity label. bltcolor() and bltcolor_r() get the ASCII color name associated with the binary label <i>label</i>.
RETURN VALUE	bltcolor() returns a pointer to a statically allocated string that contains the ASCII color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no ASCII color name is available for this binary label. bltcolor_r() returns a pointer to the <i>color_name</i> string which contains the ASCII color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no ASCII color name is available for this binary label. <i>color_name</i> must provide for a string of at least <i>size</i> characters.
FILES	/etc/security/tsol/label_encodings The label encodings file containing the <i>Classification</i> names, <i>Words</i>, constraints, and values for the defined labels of this system.
SEE ALSO	bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), hextob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL), label_encodings(4TSOL) <i>Trusted Solaris Developer's Guide</i> <i>Trusted Solaris user's document set</i> <i>Trusted Solaris administrator's document set</i>
NOTES	The function bltcolor() returns a pointer to a statically allocated string. Subsequent calls to it will overwrite that string with a new ASCII color name. It is not MT-Safe. For multithreaded applications the function bltcolor_r() should be used. If <i>label</i> includes a specified word or words, the ASCII color name associated with the first word specified in the label encodings file is returned. Otherwise, if no ASCII color name is specified for <i>label</i>, the first ASCII color name specified in the label encodings file with the same <i>Classification</i> as the binary label is returned.

These interfaces are uncommitted. Though they are not expected to change between minor releases of Trusted Solaris systems, they may.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – translate binary labels to character-coded labels
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ltsol [<i>library</i> ...]</pre> <pre>#include <tsol/label.h></pre> <pre>int bcltos(const bclabel_t *label, char **string, const int str_len, const int flags)</pre> <pre>int bsltos(const bslabel_t *label, char **string, const int str_len, const int flags)</pre> <pre>int biltos(const bilabel_t *label, char **string, const int str_len, const int flags)</pre> <pre>int bcleartos(const bclear_t *clearance, char **string, const int str_len, const int flags)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on labels that dominate the current processes' sensitivity label. These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter. The generic form of an output character-coded label is: <pre>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</pre> Capital letters are used to display all <i>Classification</i> names and <i>Words</i>. The ' ' (space) character separates <i>Classifications</i> and <i>Words</i> from other <i>Words</i> in all character-coded labels except where multiple <i>Words</i> that require the same <i>prefix</i> or <i>suffix</i> are present, in which case the multiple <i>Words</i> are separated from each other by the '/' (slash) character. <i>string</i> may point to either a pointer to pre-allocated memory, or the value (char *)0. If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value (char *)0, memory is allocated using malloc() to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory. <i>flags</i> is 0 (zero), or the logical sum of the following: LONG_WORDS Translate using long names of <i>Words</i> defined in <i>label</i>. SHORT_WORDS Translate using short names of <i>Words</i> defined in <i>label</i>. If no short name is defined in the label_encodings file for a <i>Word</i>, the long name is used. LONG_CLASSIFICATION Translate using long name of <i>Classification</i> defined in <i>label</i>. SHORT_CLASSIFICATION Translate using short name of <i>Classification</i> defined in <i>label</i>. ALL_ENTRIES Translate all entries defined in <i>Information Label label</i>.

ACCESS_RELATED

Translate only *access-related* entries defined in *Information Label label*.

VIEW_EXTERNAL

Translate *Admin Low* and *Admin High* labels to the lowest and highest labels defined in the **label_encodings** file.

VIEW_INTERNAL

Translate *Admin Low* and *Admin High* labels to the *admin low name* and *admin high name* strings specified in the **label_encodings** file. If no strings are specified, the strings “ADMIN_LOW” and “ADMIN_HIGH” are used.

NO_CLASSIFICATION

Do not translate *Classification* defined in *label*.

bcltos() translates a *Binary CMW Label* into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are **LONG_WORDS** or **SHORT_WORDS**, and **VIEW_EXTERNAL** or **VIEW_INTERNAL**. A *flags* value 0 is equivalent to (**LONG_WORDS**).

bsltos() translates a *Binary Sensitivity Label* into a string. The applicable *flags* are **LONG_CLASSIFICATION** or **SHORT_CLASSIFICATION**, **LONG_WORDS** or **SHORT_WORDS**, **VIEW_EXTERNAL** or **VIEW_INTERNAL**, and **NO_CLASSIFICATION**. A *flags* value 0 is equivalent to (**SHORT_CLASSIFICATION | LONG_WORDS**).

biltos() translates a *Binary Information Label* into a string. The applicable *flags* are **LONG_CLASSIFICATION** or **SHORT_CLASSIFICATION**, **LONG_WORDS** or **SHORT_WORDS**, **ALL_ENTRIES** or **ACCESS_RELATED**, **VIEW_EXTERNAL** or **VIEW_INTERNAL**, and **NO_CLASSIFICATION**. A *flags* value 0 is equivalent to (**LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES**).

bcleartos() translates a *Binary Clearance* into a string. The applicable *flags* are **LONG_CLASSIFICATION** or **SHORT_CLASSIFICATION**, **LONG_WORDS** or **SHORT_WORDS**, **VIEW_EXTERNAL** or **VIEW_INTERNAL**, and **NO_CLASSIFICATION**. A *flags* value 0 is equivalent to (**SHORT_CLASSIFICATION | LONG_WORDS**). The translation of a *Clearance* may not be the same as the translation of a *Sensitivity Label*. These functions use different **label_encodings** file tables that may contain different *Words* and constraints.

RETURN VALUES

These routines return:

- 1** If the label is not of the valid defined required type, if the label is not dominated by the process *Sensitivity Label* and the process does not have **PRIV_SYS_TRANS_LABEL** in its set of effective privileges, or the **label_encodings** file is inaccessible.
- 0** If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (***string[0] = '\000'**).
- >0** If successful, the length of the character-coded label including the NULL

terminator.

PROCESS ATTRIBUTES

If the **VIEW_EXTERNAL** or **VIEW_INTERNAL** flags are not specified, translation of **Admin Low** and **Admin High** labels is controlled by the label view process attribute flags. If no label view process attribute flags are defined, their translation is controlled by the label view configured in the **label_encodings** file.

A value of **External** specifies that **Admin Low** and **Admin High** labels are mapped to the lowest and highest labels defined in the **label_encodings** file.

A value of **Internal** specifies that the **Admin Low** and **Admin High** labels are translated to the **admin low name** and **admin high name** strings specified in the **label_encodings** file. If no such names are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.

FILES

/etc/security/tsol/label_encodings

The label encodings file containing the *Classification* names, *Words*, constraints, and values for the defined labels of this system.

SEE ALSO

bcltobanner(3TSOL), **bilconjoin(3TSOL)**, **blcompare(3TSOL)**, **blinset(3TSOL)**, **blmanifest(3TSOL)**, **blminmax(3TSOL)**, **blportion(3TSOL)**, **bltcolor(3TSOL)**, **bltype(3TSOL)**, **blvalid(3TSOL)**, **btohex(3TSOL)**, **free(3C)**, **hextob(3TSOL)**, **labelinfo(3TSOL)**, **labelvers(3TSOL)**, **malloc(3C)**, **sbltos(3TSOL)**, **stobl(3TSOL)**, **label_encodings(4TSOL)**

Trusted Solaris Developer's Guide

Trusted Solaris administrator's document set

NOTES

If memory is allocated by these routines, the caller must free the memory with **free()** when the memory is no longer in use.

NAME	bltype, setbltype – compare and set the type of binary label
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> int bltype(const void *label, const unsigned char type) void setbltype(void *label, const unsigned char type)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions compare and set the type of <i>BinaryLabels</i>. bltype() examines <i>label</i> to determine if it is of the specified type <i>type</i>. setbltype() sets the type of <i>label</i> to the specified type <i>type</i>. The type may be one of: SUN_SL_ID <i>label</i> is a defined <i>Binary Sensitivity Label</i>. SUN_SL_UN <i>label</i> is an undefined <i>Binary Sensitivity Label</i>. SUN_IL_ID <i>label</i> is a defined <i>Binary Information Label</i>. SUN_IL_UN <i>label</i> is an undefined <i>Binary Information Label</i>. SUN_CLR_ID <i>label</i> is a defined <i>Binary Clearance</i>. SUN_CLR_UN <i>label</i> is an undefined <i>Binary Clearance</i>. SUN_CMW_ID <i>label</i> is a <i>Binary CMW Label</i> whose <i>Sensitivity Label</i> and <i>Information Label</i> portions may or may not be defined. (bltype() only.)
RETURN VALUES	bltype() returns non-zero if <i>label</i> is of type <i>type</i> , otherwise zero is returned.
SEE ALSO	bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltocolor(3TSOL), bltos(3TSOL), blvalid(3TSOL), btohex(3TSOL), hextob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL) <i>Trusted Solaris Developer's Guide</i>
WARNINGS	Note: bltype(&cmw_label, SUN_CMW_ID) checks the existence of a <i>Binary CMW Label</i> structure and not the portions of the structure that contain defined labels. Note: When attempting to determine the type of a label, rather than to verify that a specific label type is present, SUN_CMW_ID should be checked first. Note: setbltype() makes no checks on the structure it is setting or the type value.

NAME	blvalid, bslvalid, bilvalid, bclearvalid – check validity of binary label
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> int bslvalid(const bslabel_t *label) int bilvalid(const bilabel_t *label) int bclearvalid(const bclear_t *clearance)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current processes' <i>Sensitivity Label</i>. These functions check the validity of binary labels. bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system. bilvalid() examines <i>label</i> to determine if it is a valid information label for this system. bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.
RETURN VALUE	These routines return: -1 if the <i>label_encodings</i> file is inaccessible. 0 if the binary label is not valid for this system or is not dominated by the process <i>Sensitivity Label</i> and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges, 1 if the binary label is valid for this system.
FILES	/etc/security/tsol/label_encodings The label encodings file containing the <i>Classification</i> names, <i>Words</i>, constraints, and values for the defined labels of this system.
SEE ALSO	bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltcolor(3TSOL), bltos(3TSOL), bltype(3TSOL), btohex(3TSOL), hextob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL), label_encodings(4TSOL) <i>Trusted Solaris Developer's Guide</i>
NOTES	Binary sensitivity labels are <i>valid</i> if they are contained in the SYSTEM_ACCREDITATION_RANGE as checked by blinset(3TSOL). bslvalid() is a synonym for calling blinset() with the containing set of SYSTEM_ACCREDITATION_RANGE and is included for completeness.

NAME	btohex, bcltoh, bsltoh, bilttoh, bcleartoh, bcltoh_r, bsltoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label) char *bsltoh(const bslabel_t *label) char *bilttoh(const bilabel_t *label) char *bcleartoh(const bclear_t *clearance) char *bcltoh_r(const bclabel_t *label, char *hex) char *bsltoh_r(const bslabel_t *label, char *hex) char *bilttoh_r(const bilabel_t *label, char *hex) char *bcleartoh_r(const bclear_t *clearance, char *hex) char *h_alloc(const unsigned char type); void h_free(char *hex)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe with exceptions described in NOTES .
DESCRIPTION	These functions convert <i>Binary Labels</i> into hexadecimal strings that represent the internal value. bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form: 0x<i>Information Label hexadecimal value</i> [0x<i>Sensitivity Label hexadecimal value</i>] bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form: [0x<i>Sensitivity Label hexadecimal value</i>] bilttoh() and bilttoh_r() convert a binary information label into a string of the form: 0x<i>Information Label hexadecimal value</i> bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form: 0x<i>Clearance hexadecimal value</i> h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), bilttoh_r(), and bcleartoh_r(). Valid values for <i>type</i> are: SUN_CMW_ID <i>label</i> is a binary CMW label. SUN_SL_ID <i>label</i> is a binary sensitivity label. SUN_IL_ID <i>label</i> is a binary information label.

SUN_CLR_ID *label* is a *Binary Clearance*.

h_free() frees memory allocated by **h_alloc()**.

RETURN VALUES

These functions return a pointer to a string that contains the result of the translation, or **(char *)0** if the parameter is not of the required type.

SEE ALSO

atohexlabel(1MTSOL), **hextoalabel**(1MTSOL), **bcltobanner**(3TSOL), **bilconjoin**(3TSOL), **blcompare**(3TSOL), **blinset**(3TSOL), **blmanifest**(3TSOL), **blminmax**(3TSOL), **blportion**(3TSOL), **bltcolor**(3TSOL), **bltos**(3TSOL), **bltype**(3TSOL), **blvalid**(3TSOL), **hextob**(3TSOL), **labelinfo**(3TSOL), **labelvers**(3TSOL), **sbltos**(3TSOL), **stobl**(3TSOL)

Trusted Solaris Developer's Guide

NOTES

The functions **bcltoh()**, **bsltoh()**, **biltoh()**, and **bcleartoh()** share the same statically allocated string storage. They are not MT-Safe. Subsequent calls to any of these functions will overwrite that string with the newly translated string.

For multithreaded applications the functions **bcltoh_r()**, **bsltoh_r()**, **biltoh_r()**, and **bcleartoh_r()** should be used.

NAME	clock_settime, clock_gettime, clock_getres – high-resolution clock operations
SYNOPSIS	<pre>cc [flag ...] file ... -lposix4 [library ...] #include <time.h> int clock_settime(clockid_t clock_id, const struct timespec *tp); int clock_gettime(clockid_t clock_id, struct timespec *tp); int clock_getres(clockid_t clock_id, struct timespec *res); struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* and nanoseconds */ };</pre>
MT-LEVEL	clock_gettime() is Async-Signal-Safe
DESCRIPTION	clock_settime() sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. The calling process must have the PRIV_SYS_CONFIG privilege in order to set the specified clock. clock_gettime() returns the current value <i>tp</i> for the specified clock, <i>clock_id</i>. The resolution of any clock can be obtained by calling clock_getres(). If <i>res</i> is not NULL, the resolution of the specified clock is stored in <i>res</i>. The <i>clock_id</i> for the real-time clock for the system is CLOCK_REALTIME. The values returned by clock_gettime() and specified by clock_settime() represent the amount of time (in seconds and nanoseconds) since 00:00 Universal Coordinated Time, January 1, 1970.
RETURN VALUES	clock_settime() , clock_gettime() , and clock_getres() return 0 upon success, otherwise they return -1 and set errno to indicate the error condition.
ERRORS	EINVAL <i>clock_id</i> does not specify a known clock. The <i>tp</i> argument to clock_settime() is outside the range for the given clock id. The <i>tp</i> argument to clock_settime() specified a nanosecond value less than zero or greater than or equal to 1,000,000,000. ENOSYS clock_settime(), clock_gettime(), or clock_getres() is not supported by this implementation. EPERM The requesting process does not have the PRIV_SYS_CONFIG privilege.
SUMMARY OF TRUSTED SOLARIS CHANGES	The calling process must have the PRIV_SYS_CONFIG privilege in order to set the specified clock.

SEE ALSO**time(2), ctime(3C), timer_gettime(3R)****NOTES**

Clock resolutions are implementation defined and are not settable by a process. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution.

NAME	ftw, nftw – walk a file tree																		
SYNOPSIS	For nftw(3CTSOL) only: <pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <ftw.h> int ftw(const char *path, int (*fn) (const char *, const struct stat *, int), int depth); int nftw(const char *path, int (*fn) (const char *, const struct stat *, int, struct FTW*), int depth, int flags);</pre>																		
MT-LEVEL	See the NOTES section of this page.																		
DESCRIPTION	ftw() recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, ftw() calls the user-defined function <i>fn</i>, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a stat structure (see stat(2)) containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> header, are: <table> <tr> <td>FTW_F</td><td>The object is a file.</td></tr> <tr> <td>FTW_D</td><td>The object is a directory.</td></tr> <tr> <td>FTW_DNR</td><td>The object is a directory that cannot be read. Descendants of the directory will not be processed.</td></tr> <tr> <td>FTW_NS</td><td>stat failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The stat buffer passed to <i>fn</i> is undefined.</td></tr> </table> ftw() visits a directory before visiting any of its descendants. The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a nonzero value, or some error is detected within ftw() (such as an I/O error). If the tree is exhausted, ftw() returns zero. If <i>fn</i> returns a nonzero value, ftw() stops its tree traversal and returns whatever value was returned by <i>fn</i>. nftw(3CTSOL) recursively descends the directory hierarchy rooted in <i>path</i>. The <i>flags</i> argument is used to control the tree walk and holds one of these values: <table> <tr> <td>FTW_PHYS</td><td>Physical walk, does not follow symbolic links. Otherwise, nftw() will follow links but will not walk down any path that crosses itself.</td></tr> <tr> <td>FTW_MOUNT</td><td>The walk will not cross a mount point.</td></tr> <tr> <td>FTW_DEPTH</td><td>All subdirectories will be visited before the directory itself.</td></tr> <tr> <td>FTW_CHDIR</td><td>The walk will change to each directory before reading it.</td></tr> <tr> <td>FTW_TSOL_MLD</td><td>In all multilevel directories (MLDs) encountered as nftw walks the tree, the walk will visit single-level directories (SLDs) that are dominated by the sensitivity label of the process if the process is run without privilege.</td></tr> </table>	FTW_F	The object is a file.	FTW_D	The object is a directory.	FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.	FTW_NS	stat failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The stat buffer passed to <i>fn</i> is undefined.	FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, nftw() will follow links but will not walk down any path that crosses itself.	FTW_MOUNT	The walk will not cross a mount point.	FTW_DEPTH	All subdirectories will be visited before the directory itself.	FTW_CHDIR	The walk will change to each directory before reading it.	FTW_TSOL_MLD	In all multilevel directories (MLDs) encountered as nftw walks the tree, the walk will visit single-level directories (SLDs) that are dominated by the sensitivity label of the process if the process is run without privilege.
FTW_F	The object is a file.																		
FTW_D	The object is a directory.																		
FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.																		
FTW_NS	stat failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The stat buffer passed to <i>fn</i> is undefined.																		
FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, nftw() will follow links but will not walk down any path that crosses itself.																		
FTW_MOUNT	The walk will not cross a mount point.																		
FTW_DEPTH	All subdirectories will be visited before the directory itself.																		
FTW_CHDIR	The walk will change to each directory before reading it.																		
FTW_TSOL_MLD	In all multilevel directories (MLDs) encountered as nftw walks the tree, the walk will visit single-level directories (SLDs) that are dominated by the sensitivity label of the process if the process is run without privilege.																		

If the effective privilege set of the process includes the **PRIV_FILE_MAC_READ** and **PRIV_FILE_MAC_SEARCH** privileges, the walk visits all SLDs in each MLD. The file system enforces all underlying DAC policies and privilege interpretations.

If the **FTW_TSOL_MLD** flag is *not* specified and *path* points to an adorned MLD, the walk traverses only the SLDs of this MLD. All other MLDs encountered while walking down the tree are automatically translated to the SLD at the sensitivity label of the process even if the process is run with all privileges.

If the **FTW_TSOL_MLD** flag is *not* specified and *path* points to an unadorned MLD, when the walk down the tree encounters this and all other MLDs, then the function automatically translates to the SLD at the sensitivity label of the process.

If the **FTW_TSOL_MLD** flag is *not* specified and *path* does not point to an MLD, when the walk down the tree encounters any MLDs, then the function automatically translates to the SLD at the sensitivity label of the process even if the process is run with all privileges.

For each object in the hierarchy, **nftw** calls the user-defined function *fn* with four arguments. The first argument is a pointer to a null-terminated character string containing the pathname of the object, the second is a pointer to the **stat** structure [see **#ifdef TSOL stat(2TSOL)**] **#else stat(2)**] **#endif** containing information about the object, the third is an integer giving additional information, and the fourth is a **struct FTW** that contains these members:

```
int base;
int level;
```

base is the offset into the pathname of the base name of the object. **level** indicates the depth relative to the rest of the walk, where the root level is zero.

The values of the third argument are as follows:

FTW_F	The object is a file.
FTW_D	The object is a directory.
FTW_DP	The object is a directory and subdirectories have been visited.
FTW_SL	The object is a symbolic link.
FTW_SLN	The object is a symbolic link that points to a non-existent file.
FTW_DNR	The object is a directory that cannot be read. <i>fn</i> will not be called for any of its descendants.
FTW_NS	stat failed on the object because of lack of appropriate permission. The stat buffer passed to <i>fn</i> is undefined. stat failure other than lack of appropriate permission. EACCES is considered an error and nftw() will return -1 .

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error (such as an I/O error) is detected within **nftw**. If the tree is exhausted, **nftw** returns zero. If *fn* returns a nonzero value, **nftw** stops its tree traversal

and returns whatever value *fn* returned.

Both **ftw()** and **nftw()** use one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. **ftw()** will run faster if *depth* is at least as large as the number of levels in the tree. When **ftw()** and **nftw()** return, they close any file descriptors they have opened; they do not close any file descriptors that may have been opened by *fn*.

RETURN VALUES

If successful, **ftw()** and **nftw()** return **0**. If either function detects an error other than **EACCES**, it returns **-1**, and sets the error type in **errno**.

SUMMARY OF TRUSTED SOLARIS CHANGES

The **libc** versions of **ftw** and **nftw** are unchanged. The Trusted Solaris version of **nftw**, which has the additional flag **FTW_TSOL_MLD**, is available in **libtsol**. You must be careful of the library sequence when linking.

SEE ALSO

stat(2TSOL), **longjmp(3C)**, **malloc(3C)**

NOTES

Because **ftw()** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

ftw() uses **malloc(3C)** to allocate dynamic storage during its operation. If **ftw()** is forcibly terminated, such as by **longjmp(3C)** being executed by *fn* or an interrupt routine, **ftw()** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

ftw() is safe in multi-thread applications. **nftw()** is safe in multi-thread applications when the **FTW_CHDIR** flag is not set.

There are two versions of **nftw**. The Solaris version, which does not traverse multilevel directories (MLDs), is located in **libc**; the Trusted Solaris version, which traverses MLDs, is located in **libtsol**. To use the Trusted Solaris version of **nftw**, make sure that the application uses the version of **nftw** located in **libtsol**.

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get information about audit-control file
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsi -lintl [<i>library</i> ...]</pre> <pre>#include <bsm/libbsm.h></pre> <pre>int getacdir(char *<i>dir</i>, int <i>len</i>);</pre> <pre>int getacmin(int *<i>min_val</i>);</pre> <pre>int getacflg(char *<i>auditstring</i>, int <i>len</i>);</pre> <pre>int getacna(char *<i>auditstring</i>, int <i>len</i>);</pre> <pre>void setac(void);</pre> <pre>void endac(void);</pre>
MT-LEVEL	Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	When first called, getacdir provides information about the first audit directory in the audit_control file; thereafter, getacdir returns the next directory in the file. Successive calls list all the directories listed in audit_control(4TSOL). The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry. getacmin reads the minimum value from the audit_control file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script audit_warn(1MTSOL) is invoked. getacflg reads the system audit value from the audit_control file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. getacna reads the system audit value for nonattributable audit events from the audit_control file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Nonattributable events are events that cannot be attributed to an individual user. inetd(1MTSOL) and several other daemons record nonattributable events. Calling setac rewinds the audit_control file to allow repeated searches. Calling endac closes the audit_control file when processing is complete.
RETURN VALUES	These functions return one of these values: 0 Success 1 EOF 2 The directory search had to start from the beginning because one of the other functions was called between calls to getacdir. -1 EOF

- 2 Failure (**errno** indicates the error.)
- 3 The directory entry in the **audit_control** file is incorrect or the input buffer is too short to accommodate the record.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

By default, the audit module is enabled on Trusted Solaris systems.

FILES **/etc/security/audit_control** Contains default parameters read by the audit daemon, **auditd**(1MTSOL)

SEE ALSO **audit_warn**(1MTSOL), **inetd**(1MTSOL), **audit_control**(4TSOL) *Trusted Solaris Audit Administration Manual*

NOTES This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing.

NAME	getauclassnam, getauclassent, setauclass, endauclass, getauclassnam_r, getauclassent_r – Get audit-class entry
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...]</pre> <pre>#include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char *name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char *name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
AVAILABILITY	SUNWcsu
MT-LEVEL	MT-Safe with exceptions All of the functions described in this man-page are MT-Safe except getauclassent() and getauclassnam(). The two functions, getauclassent_r() and getauclassnam_r() have the same functionality as the unsafe functions but have a slightly different function-call interface in order to make them MT-Safe.
DESCRIPTION	getauclassent() and getauclassnam() each return an audit-class entry. getauclassnam() searches for an audit-class entry with a given class name <i>name</i>. getauclassent() lists audit_class entries; successive calls to getauclassent will return either successive audit-class entries or NULL. setauclass() “rewinds” to the beginning of the list of audit-class entries. Because calls to getauclassnam may leave the list in an indeterminate state, setauclass() should be called before the first getauclassent. endauclass() may be called to indicate that audit-class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth. getauclassent_r() and getauclassnam_r() return a pointer to an audit_class entry as do their similarly named counterparts. getauclassent_r and getauclassnam_r each take an additional argument, a pointer (<i>class_int</i>) to preallocated space for an au_class_ent_t, which is returned if the call is successful. To ensure that there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the <i>ac_name</i> and <i>ac_desc</i> elements of the au_class_ent_t data structure in <bsm/libbsm.h>.

The internal representation of an audit-user entry is an **au_class_ent** structure defined in **<bsm/libbsm.h>** with these members:

```
char    *ac_name;  
au_class_t  ac_class;  
char    *ac_desc;
```

RETURN VALUES

getaclassnam and **getaclassnam_r** return a pointer to a **struct au_class_ent** if they successfully locate the requested entry; otherwise they return NULL.

getaclassent and **getaclassent_r** return a pointer to a **struct au_class_ent** if they successfully list an entry; otherwise they return NULL, indicating the end of the list.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

By default, the audit module is enabled on Trusted Solaris systems.

FILES

/etc/security/audit_class Maps audit-class numbers to audit-class names

SEE ALSO

audit_class(4TSOL), **audit_event**(4TSOL), *Trusted Solaris Audit Administration Manual*

NOTES

This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing.

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit-flag specifications
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char *auditstring, au_mask_t *masks); int getauditflagschar(char *auditstring, au_mask_t *masks, int verbose);</pre>
MT-LEVEL	MT-Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	getauditflagsbin converts the character representation of audit values to which <i>auditstring</i> points into au_mask_t fields to which <i>masks</i> point. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The syntax of the character string is described in audit_control(4TSOL). getauditflagschar converts the au_mask_t fields to which <i>masks</i> point into a string to which <i>auditstring</i> points. If <i>verbose</i> is zero, the short (2-character) flag names are used; if <i>verbose</i> is nonzero, the long flag names are used. <i>auditstring</i> should be large enough to contain the ASCII representation of the events. <i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The au_mask_t fields to which <i>masks</i> points correspond to binary values defined in <bsm/audit.h>, which is read by <bsm/libbsm.h>.
RETURN VALUES	Upon success, getauditflagsbin and getauditflagschar return 0 . Upon failure, they return -1 .
SUMMARY OF TRUSTED SOLARIS CHANGES	The getauditflagsbin and getauditflagschar functions replace getauditflags. By default, the audit module is enabled on Trusted Solaris systems.
SEE ALSO	audit.log (4TSOL), audit_control (4TSOL) <i>Trusted Solaris Audit Administration Manual</i>
NOTES	This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for how to disable and enable auditing.
BUGS	This is not a very extensible interface.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endaeuevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_user entry
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns -lintl [<i>library</i> ...]</pre> <pre>#include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *<i>name</i>); struct au_event_ent *getauevnum(au_event_t <i>event_number</i>); au_event_t *getauevnonam(char *<i>event_name</i>); void setauevent(void); void endaeuevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *<i>e</i>); struct au_event_ent *getauevnam_r(au_event_ent_t *<i>e</i>, char *<i>name</i>); struct au_event_ent *getauevnum_r(au_event_ent_t *<i>e</i>, au_event_t <i>event_number</i>);</pre>
MT-LEVEL	MT-Safe with exceptions The functions getauevent, getauevnam, and getauevnum are not MT-Safe; but there are equivalent functions that provide the same functionality and a MT-Safe function-call interface: getauevent_r, getauevnam_r, and getauevnum_r.
AVAILABILITY	SUNWcsu
DESCRIPTION	getauevent, getauevnam, and getauevnum, each return a pointer to an audit_event structure. getauevent and getauevent_r list audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL. getauevnam and getauevnam_r search for an audit_event entry with a given event name <i>name</i>. getauevnum and getauevnum_r search for an audit_event entry with a given event number <i>number</i>. getauevnonum searches for an audit_event entry with a given event name <i>name</i> and returns the corresponding event number. setauevent “rewinds” to the beginning of the list of audit_event entries. Because calls to getauevnam, getauevnum, getauevnonum, getauevnam_r, or getauevnum_r may leave the list in an indeterminate state, setauevent should be called before the first getauevent or getauevent_r. endaeuevent may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.

The three functions **getauevent_r**, **getauevnam_r**, and **getauevnum_r** each take an argument *e*, which is a pointer to an **au_event_ent_t**. This pointer is returned on a successful function call. To ensure that there is enough space for the information returned, the applications programmer should allocate **AU_EVENT_NAME_MAX** and **AU_EVENT_DESC_MAX** bytes for the **ae_name** and **ae_desc** elements of the **au_event_ent_t** data structure.

The internal representation of an **audit_event** entry is a **struct au_event_ent** structure defined in **<bsm/libbsm.h>** with these members:

```

au_event_t  ae_number;
char        *ae_name;
char        *ae_desc;
au_class_t  ae_class;

```

RETURN VALUES

If they successfully locate the requested entry **getauevent_r**, **getauevnam_r**, **getauevnum_r**, **getauevent**, **getauevnam** and **getauevnum** return a pointer to a **struct au_event_ent**. Otherwise, they return **NULL**.

If it successfully lists an entry, **getauevnonam** returns an event number of type **au_event_t**. Otherwise if it cannot find the requested event name, the function returns **NULL**.

FILES

/etc/security/audit_event	Maps audit-event numbers to audit-event names
/etc/passwd	Stores mappings of user ID to user name

SUMMARY OF TRUSTED SOLARIS CHANGES

By default, the audit module is enabled on Trusted Solaris systems.

SEE ALSO

getpwnam(3C), **getauctlassent(3TSOL)**, **audit_class(4TSOL)**, **audit_event(4TSOL)**, **passwd(4)** *Trusted Solaris Audit Administration Manual*

NOTES

This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing. Because information for the functions **getauevent**, **getauevnam**, and **getauevnum** is contained in a static area, which may be overwritten, all information must be copied if it is to be saved.

NAME	getauusernam, getauuserent, getauusernam_r, getauuserent_r, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...]</pre> <pre>#include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent *getauusernam(const char *name); struct au_user_ent *getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent *getauusernam_r(au_user_ent *u, const char *name); struct au_user_ent *getauuserent_r(au_user_ent *u);</pre>
AVAILABILITY	SUNWcsu
MT-LEVEL	MT-Safe with exceptions The functions getauusernam and getauuserent are not MT-safe. However, the functions getauusernam_r and getauuserent_r provide the same functionality with an MT-Safe interfaces.
DESCRIPTION	getauuserent, getauusernam, getauuserent_r, and getauusernam_r each return an audit_user entry. getauusernam and getauusernam_r search for an audit_user entry with a given login name <i>name</i>. getauuserent and getauuserent_r list audit_user entries; successive calls to these functions will return either successive audit_user entries or NULL. setauuser “rewinds” to the beginning of the list of audit_user entries. Because calls to getauusernam and getauusernam_r may leave the list in an indeterminate state, setauuser should be called before the first getauuserent or getauuserent_r. endauuser may be called to indicate that audit_user processing is complete; the system may then close any open audit_user file, deallocate storage, and so forth. The two functions getauuserent_r and getauusernam_r both take an argument <i>u</i>, which is a pointer to an au_user_ent. This pointer is returned on successful function calls. The internal representation of an audit_user entry in an au_user_ent structure, defined in <bsm/libbsm.h>, has the following members: <pre>char *au_name; au_mask_t au_always; au_mask_t au_never;</pre>

RETURN VALUES

If they successfully locate the requested entry, **getauusernam_r**, and **getauusernam** return a pointer to a **struct au_user_ent**. Otherwise they return NULL.

If they successfully list an entry, **getauuserent_r**, and **getauuserent** return a pointer to a **struct au_user_ent**. Otherwise they return NULL, marking the end of the list.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

By default, the audit module is enabled on Trusted Solaris systems.

FILES

/etc/security/audit_user	Stores per-user audit-event mask
/etc/passwd	Stores mappings of user ID to user name

SEE ALSO

getpwnam(3C), **audit_user(4TSOL)**, **passwd(4)** *Trusted Solaris Audit Administration Manual*

NOTES

This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing.

All information for the functions **getauuserent** and **getauusernam** is contained in a static area, which may be overwritten, so the information must be copied if it is to be saved.

NAME	getfauditflags – Generates the process audit state
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsi -lintl [<i>library</i> ...]</pre> <pre>#include <sys/param.h> #include <bsm/libbsm.h> int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);</pre>
MT-LEVEL	MT-Safe
AVAILABILITY	SUNWcsu
DESCRIPTION	The auditing module required to run this command is enabled by default in the Trusted Solaris system. Trusted Solaris 2.x has extended the audit class and mask structures from 32 to 64 classes with the option to extend further if necessary. Therefore, applications should be using the extended audit-class variants of any call identified by the _x suffix. getfauditflags generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the audit_control(4TSOL) file. getfauditflags obtains the system audit value by calling getacflg. [See getacinfo(3TSOL).] <i>usremasks</i> points to au_mask_t fields that contain two values. The first value defines which events are <i>always</i> to be audited when they succeed. The second value defines which events are <i>always</i> to be audited when they fail. <i>usrdmasks</i> also points to au_mask_t fields that contain two values. The first value defines which events are <i>never</i> to be audited when they succeed. The second value defines which events are <i>never</i> to be audited when they fail. The structures pointed to by <i>usremasks</i> and <i>usrdmasks</i> may be obtained from the audit_user(4TSOL) file by calling getauusernam or getauusernam to return a pointer to a structure containing all audit_user(4TSOL) fields for a user. The output of this function is stored in <i>lastmasks</i>, which is a pointer of type au_mask_t as well. The first value defines which events are to be audited when they succeed; the second defines which events are to be audited when they fail. Both <i>usremasks</i> and <i>usrdmasks</i> override the values in the system audit values.
RETURN VALUES	Upon success, getfauditflags returns 0 . Upon failure, getfauditflags returns -1 .
SUMMARY OF TRUSTED SOLARIS CHANGES	By default, the audit module is enabled on Trusted Solaris systems.
SEE ALSO	getacinfo (3TSOL), getauditflags (3TSOL), getauusernam (3TSOL), audit.log (4TSOL), audit_control (4TSOL), audit_user (4TSOL)

NOTES

This functionality is active only if the audit module has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for how to disable and enable auditing.

NAME	getpeerinfo – get peer’s process characteristics.
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsi -lintl [<i>library</i> ...]</pre> #include <bsm/audit.h> <pre>int getpeerinfo(int <i>fd</i>, au_peergroupinfo_t *<i>grpinfo</i>, au_peermiscinfo_t *<i>prinfo</i>);</pre>
MT_LEVEL	MT-Safe.
AVAILABILITY	Available only on Trusted Solaris systems with the auditing module enabled. The auditing module is enabled by default in the Trusted Solaris system.
DESCRIPTION	Returns the peer process’ audit attributes for the peer designated by the socket or TLI file descriptor <i>fd</i>. If <i>grpinfo</i> or <i>peerinfo</i> is NULL, then the corresponding information is not obtained. The au_peergroupinfo structure has the following form: <pre>struct au_peergroupinfo { ulong_t peer_ngroups /* number of elements obtained */ gid_t peer_groups[NGROUPS_UMAX]; /* peer’s supplemental groups */ };</pre> The remaining attributes are returned in <i>prinfo</i> which is of type struct au_peermiscinfo and has been allocated by the calling process. The au_peermiscinfo structure has the following form: <pre>struct au_peermiscinfo { uid_t peer_ruid; /* peer’s real user id */ gid_t peer_rgid; /* peer’s real group id */ auditinfo_t peer_audit; /* peer’s audit characteristic’s */ };</pre> where auditinfo_t is of type struct auditinfo which has the following form: <pre>struct auditinfo { au_id_t ai_auid; /* audit ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* audit terminal ID */ au_asid_t ai_asid; /* audit session ID */ };</pre> getpeerinfo() requires that either the PRIV_PROC_AUDIT_TCB or PRIV_PROC_AUDIT_APPL privilege be asserted in a process’ effective set in order to get the prinfo attributes from its peer. No privileges are required to obtain just grpinfo.

RETURN VALUES

getpeerinfo() returns **0** on success. On failure it returns a negative value and sets **errno** to indicate the error.

ERRORS

EBADF	<i>fd</i> is not a valid descriptor.
ENOTSOCK	<i>fd</i> is not a socket or TLI interface.
ENOBUFS	Insufficient resources were available in the system to perform the operation.
EADDRNOTAVAIL	Could not establish connection with server.
EINVAL	There was an internal error in which <i>fd</i> pointed to a peer process that was not recognized by its host.
ENOENT	No such port currently active on the peer.
EOPNOTSUPP	Type not SOCK_DGRAM or SOCK_STREAM , or either the local peer socket or TLI descriptor is not AF_INET .
EPERM	The caller does not have the proper privileges.

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – get Trusted Solaris 2.x user profile description
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol -ltsol -lnsl -lcmd [library ...] #include <tsol/prof.h> profent_t *getprofentbyname(char *name, int src); profent_t *getprofent(int src); void setprofent(int stayopen, int src); void endprofent(int src); void free_profent(profent_t *profent);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file. getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i>. The functions setprofent(), getprofent(), and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent(). A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent(). Successive calls to getprofent() return either successive entries or return NULL, indicating the end of the enumeration. endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent(). The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent(). The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string. The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES, TSOL_DB_SRC_NISPLUS, or TSOL_DB_SRC_SWITCH, which are defined in <tsol/tsol.h>. For most applications the <i>src</i> parameter should be set to TSOL_DB_SRC_SWITCH, indicating that the system should use the /etc/nsswitch.conf file to determine the ultimate source of the database.

However, in certain administrative application, it may be prudent to use the **TSOL_DB_SRC_FILES** option to for a read from the **/etc/security/tsol/tsoluser** file or the **TSOL_DB_SRC_NISPLUS** option to force a read from the tsoluser NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setprofent()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getprofent()** , the threads will enumerate disjoint subsets of the **tsolprof database**.

RETURN VALUES

User entries are represented by the **struct profent_t** structure defined in **<tsol/prof.h>** :

```
typedef struct profent_s {
    char *      name;      /* name of profile */
    char *      desc;      /* description */
    char *      auths;     /* comma separated list of authorization numbers */
    profact_t * actions;   /* linked list of actions */
    profcmd_t * cmds;      /* linked list of commands */
} profent_t;
```

The function **getprofentbyname()** returns a pointer to a **profent_t** if it successfully locates the requested entry; otherwise it returns **NULL** .

The function **getprofent()** returns a pointer to a **struct profent_t** if it successfully enumerates an entry; otherwise it returns **NULL** , indicating the end of the enumeration.

ERRORS

The functions **getprofentbyname()** and **getprofent()** return **NULL** on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see **Intro (3), Notes On Multithread Applications**, for information about the use of the **_REENTRANT** flag.

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – get Trusted Solaris 2.x user profile description
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ltsol -ltsol -lnsl -lcmd [<i>library</i> ...]</pre> <pre>#include <tsol/prof.h></pre> <pre>profstr_t *getprofstrbyname(char *name, int src);</pre> <pre>profstr_t *getprofstr(int src);</pre> <pre>int putprofstr(profstr_t *res, int src);</pre> <pre>int setprofstr(int stayopen, int src);</pre> <pre>int endprofstr(int src);</pre> <pre>void free_profstr(profstr_t *profstr);</pre>
MT-LEVEL	MT-Safe
AVAILABILITY	SUNWtsolu
DESCRIPTION	These functions are used to obtain entries describing Trusted Solaris 2.x user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file. getprofbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i>. The functions setprofstr(), getprofstr(), and endprofstr() are used to enumerate profile entries from the database. setprofstr() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofstr(). A call to getprofbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofstr(). Successive calls to getprofstr() return either successive entries or return NULL, indicating the end of the enumeration. endprofstr() may be called to indicate that the caller expects to do no further profile strry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile strry retrieval functions after calling endprofstr(). The functions getprofstrbyname() and getprofstr() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profstr() should be used to free the pointers returned by either getprofstrbyname() or getprofstr(). The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.

The parameter *src* may be set to any of **TSOL_SRC_FILES**, **TSOL_SRC_NISPLUS**, or **TSOL_SRC_SWITCH**, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to **TSOL_SRC_SWITCH**, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the **TSOL_SRC_FILES** option to for a read from the `/etc/security/tsol/tsoluser` file or the **TSOL_SRC_NISPLUS** option to force a read from the tsoluser NIS+ database.

The function **putprofstr** replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter treated as **TSOL_SRC_NISPLUS**, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf(4)` may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setprofstr()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getprofstr()**, the threads enumerate disjoint subsets of the **tsolprof(4TSOL)** database.

RETURN VALUES

User entries are represented by the **struct profstr_t** structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_s {
    char *   name;      /* name of profile */
    char *   desc;      /* description */
    char *   auths;     /* comma separated list of authorization numbers */
    char *   actions;   /* semicolon separated action descriptions */
    char *   cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

The function **getprofstrbyname()** returns a pointer to a **profstr_t** if it successfully locates the requested entry; otherwise it returns **NULL**.

The function **getprofstr()** returns a pointer to a **struct profstr_t** if it successfully enumerates an entry; otherwise it returns **NULL**, indicating the end of the enumeration.

The function **putprofstr()** return **0** on success.

ERRORS

The functions **getprofstrbyname()** and **getprofstr()** return **NULL** on on a failure.

NOTES

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see **Intro (3), Notes On Multithread Applications**, for information about the use of the **_REENTRANT** flag.

NAME	getsockopt, setsockopt – get and set options on sockets														
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> int getsockopt(int s, int level, int optname, char *optval, int *optlen); int setsockopt(int s, int level, int optname, const char *optval, int optlen);</pre>														
MT-LEVEL	Safe														
DESCRIPTION	getsockopt() and setsockopt() manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost “socket” level. When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the “socket” level, <i>level</i> is specified as SOL_SOCKET. To manipulate options at any other level, <i>level</i> is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, <i>level</i> is set to the TCP protocol number (see getprotobyname(3N)). The parameters <i>optval</i> and <i>optlen</i> are used to access option values for setsockopt(). For getsockopt(), they identify a buffer in which the value(s) for the requested option(s) are to be returned. For getsockopt(), <i>optlen</i> is a value-result parameter, initially containing the size of the buffer pointed to by <i>optval</i>, and modified on return to indicate the actual size of the value returned. Use a 0 <i>optval</i> if no option value is to be supplied or returned. <i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name. Most socket-level options take an int for <i>optval</i>. For setsockopt(), the <i>optval</i> parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter that specifies the desired state of the option and the linger interval (see below). struct linger is defined in <sys/socket.h>. struct linger contains the following members: <table> <tr> <td>l_onoff</td><td>on = 1 / off = 0</td></tr> <tr> <td>l_linger</td><td>linger time, in seconds</td></tr> </table> The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt(). <table> <tr> <td>SO_DEBUG</td><td>enable / disable recording of debugging information</td></tr> <tr> <td>SO_REUSEADDR</td><td>enable / disable local address reuse</td></tr> <tr> <td>SO_KEEPAIVE</td><td>enable / disable keep connections alive</td></tr> <tr> <td>SO_DONTROUTE</td><td>enable / disable routing bypass for outgoing messages</td></tr> <tr> <td>SO_LINGER</td><td>linger on close if data is present</td></tr> </table>	l_onoff	on = 1 / off = 0	l_linger	linger time, in seconds	SO_DEBUG	enable / disable recording of debugging information	SO_REUSEADDR	enable / disable local address reuse	SO_KEEPAIVE	enable / disable keep connections alive	SO_DONTROUTE	enable / disable routing bypass for outgoing messages	SO_LINGER	linger on close if data is present
l_onoff	on = 1 / off = 0														
l_linger	linger time, in seconds														
SO_DEBUG	enable / disable recording of debugging information														
SO_REUSEADDR	enable / disable local address reuse														
SO_KEEPAIVE	enable / disable keep connections alive														
SO_DONTROUTE	enable / disable routing bypass for outgoing messages														
SO_LINGER	linger on close if data is present														

SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_OOBINLINE	enable/disable reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a **bind(3NTSOL)** call should allow reuse of local addresses. **SO_KEEPAIVE** enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a **SIGPIPE** signal. **SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on a socket and a **close(2)** is performed. If the socket promises reliable delivery of data and **SO_LINGER** is set, the system will block the process on the **close()** attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the **setsockopt()** call when **SO_LINGER** is requested). If **SO_LINGER** is disabled and a **close()** is issued, the system will process the **close()** in a manner that allows the process to continue as quickly as possible.

The option **SO_BROADCAST** requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the **SO_OOBINLINE** option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with **recv()** or **read()** calls without the **MSG_OOB** flag. No privilege is required to set the **SO_BROADCAST** flag, and any user may do so; however, the **PRIV_NET_BROADCAST** privilege is required to use a broadcast address.

SO_SNDBUF and **SO_RCVBUF** are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. The Internet protocols place an absolute limit of 64 Kbytes on these values for **UDP** and **TCP** sockets.

Finally, **SO_TYPE** and **SO_ERROR** are options used only with **getsockopt()**. **SO_TYPE** returns the type of the socket (for example, **SOCK_STREAM**). It is useful for servers that inherit sockets on startup. **SO_ERROR** returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

SUMMARY OF TRUSTED SOLARIS CHANGES

A process must have the **PRIV_NET_RAWACCESS** privilege in order to specify IP options 130 or 134 (**IPOPT_SEC** and **IPOPT_CIPSO**, respectively, as defined in **<inet/ip.h>**). The former refers to the Basic Security Option and the latter refers to the CIPSO option. A process must have the **PRIV_NET_BROADCAST** privilege to use a broadcast address.

RETURN VALUES	If successful, getsockopt() returns 0 ; otherwise, it returns -1 and sets errno to indicate the error.
ERRORS	The call succeeds unless: EBADF The argument <i>s</i> is not a valid file descriptor. ENOMEM There was insufficient memory available for the operation to complete. ENOPROTOOPT The option is unknown at the level indicated or the specified option requires PRIV_NET_RAWACCESS privilege. ENOSR There were insufficient STREAMS resources available for the operation to complete. ENOTSOCK The argument <i>s</i> is not a socket.
SEE ALSO	close(2) , ioctl(2) , bind(3NTSOL) , getprotobyname(3N) , socket(3N)

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol -ltsol -lnsl -lcmd [library ...] #include <tsol/user.h> userent_t *getuserentbyname(char *user, int src); userent_t *getuserentbyuid(uid_t uid, int src); userent_t *getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t *userent);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions are used to obtain entries describing Trusted Solaris user attributes from the tsoluser NIS+ database. getuserbyname() searches for information for a user with the specified user name <i>user</i>. getuserbyuid() searches for information for a user with the specified user ID <i>uid</i>. The functions setuserent(), getuserent(), and enduserent() are used to list user entries from the database. setuserent sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to getuserent. A call to getuserbyname() or getuserbyuid() leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to enduserent. Successive calls to getuserent return either successive entries or NULL, which indicates the end of the list. enduserent may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling enduserent. The functions getuserentbyname, getuserentbyuid, and getuserent are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_userent() is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a uid_t. The parameter <i>src</i> may be set to TSOL_DB_SRC_FILES, TSOL_DB_SRC_NISPLUS, or TSOL_DB_SRC_SWITCH, which are defined in <tsol/tsol.h>.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. **setuserent** may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to **getuserent**, the threads will list disjoint subsets of the **tsoluser** database.

RETURN VALUES

User entries are represented by the **struct userent_t** structure defined in `<tsol/user.h>` :

```
typedef struct userent_s {
    char *   name;           /* user associated with this entry */
    char *   lock;           /* is account locked? */
    char *   badlogins;      /* how many failed login attempts so far */
    char *   generation;     /* method of password generation */
    char *   profiles;       /* user profiles used */
    char *   roles;          /* roles assumable */
    char *   idletime;        /* minutes a workstation may remain idle */
    char *   idlcmd;         /* what to do at when idletime reached */
    char *   labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *   labeltrans;     /* process security attributes for label translation */
    char *   labelmin;       /* allowed low login labels */
    char *   labelmax;       /* allowed high login labels */
    char *   usertype;       /* normal, admin-role, or non-admin-role */
    char *   res1;           /* reserved for future use */
    char *   res2;           /* reserved for future use */
    char *   res3;           /* reserved for future use */
} userent_t;
```

If it successfully locates the requested entry, **getuserentbyname** returns a pointer to a **userent_t**. If unsuccessful, **getuserentbyname** returns **NULL**.

If it successfully lists an entry, **getuserent** returns a pointer to a **struct userent_t**. If unsuccessful, **getuserent** returns **NULL**, indicating the end of the list.

Upon success, **setuserent** and **enduserent** return **0**.

ERRORS

The functions **getuserentbyname**, **getuserentbyuid**, and **getuserent** return **NULL** on failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see *Notes on Multithread Applications* in **Intro(3)** for information about the use of the **_REENTRANT** flag.

These interfaces are uncommitted; although not expected to change between minor releases of Trusted Solaris operating environments, these interfaces may change.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	getutent(), getutid(), getutline(), and pututline() each return a pointer to a utmp structure with the following members: <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id */ /* (usually line #) */ char ut_line[12]; /* device name (console, lxxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> The structure exit status includes the following members: <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> getutent() reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails. getutid() searches forward from the current point in the utmp file until it finds an entry with a ut_type matching <i>id</i>→ut_type if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in <i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then getutid() will return a pointer to the first entry whose type is one of these four and whose ut_id field matches <i>id</i>→ut_id. If the end of file is reached without a match, it fails. getutline() searches forward from the current point in the utmp file until it finds an entry of the type LOGIN_PROCESS or ut_line string matching the <i>line</i>→ut_line string. If the end of file is reached without a match, it fails. pututline() writes out the supplied utmp structure into the utmp file. It uses getutid() to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of pututline() will have searched for the proper entry

using one of the these routines. If so, **pututline()** will not search. If **pututline()** does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the **utmp** structure. When called by a process that does not have an effective uid of **0** and a sensitivity label of **ADMIN_LOW**, **pututline()** invokes a program (that has the appropriate forced privileges) to verify and write the entry, since **/etc/utmpx** is normally writable only by a process with a UID of **0** and a sensitivity label of **ADMIN_LOW**. In this event, the **ut_name** field must correspond to the actual user name associated with the process; the **ut_type** field must be either **USER_PROCESS** or **DEAD_PROCESS**; and the **ut_line** field must be a device special file and be writable by the user. If the process does not have the **PAF_TRUSTED_PATH** process attribute, all other fields in the entry are cleared.

setutent() resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() closes the currently open file.

utmpname() allows the user to change the name of the file examined, from **/var/adm/utmp** to any other file. It is most often expected that this other file will be **/var/adm/wtmp**. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. **utmpname()** does not open the file. It just closes the old file if it is currently open and saves the new file name.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, **utmpname()** returns 0. Otherwise, it returns 1.

FILES

/var/adm/utmp
/var/adm/wtmp

SEE ALSO

getutxent(3C), **ttyslot(3C)**, **utmp(4)**

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either **getutid()** or **getutline()**, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use **getutline()** to search for multiple occurrences, it would be necessary to zero out the static area after each success, or **getutline()** would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by **pututline()** (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the **getutent()**, **getutid()** or **getutline()** routines, if the user has just modified those contents and passed the pointer back to **pututline()**.

These routines use buffered standard I/O for input, but **pututline()** uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the **utmp** and **wtmp**

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

pututline() invokes a program with appropriate forced privileges to verify and write the **utmpx** structure. **pututline** clears fields in an entry if the process does not have the PAF_TRUSTED_PATH process attribute.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure with the following members: <pre> char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id */ /* (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> The structure exit status includes the following members: <pre> short e_termination; /* termination status */ short e_exit; /* exit status */</pre> getutxent() Reads in the next entry from a utmpx-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails. getutxid() Searches forward from the current point in the utmpx file until it finds an entry with a ut_type matching <i>id</i>→ut_type if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in <i>id</i> is INIT_PROCESS, LOGIN_PROCESS,

	USER_PROCESS , or DEAD_PROCESS , then getutxid() will return a pointer to the first entry whose type is one of these four and whose <i>ut_id</i> field matches <i>id</i> → ut_id . If the end of file is reached without a match, it fails.
getutxline()	Searches forward from the current point in the utmpx file until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a <i>ut_line</i> string matching the <i>line</i> → ut_line string. If the end of file is reached without a match, it fails.
pututxline()	Writes out the supplied utmpx structure into the utmpx file. It uses getutxid() to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of pututxline() will have searched for the proper entry using one of the getutx() routines. If so, pututxline() will not search. If pututxline() does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the utmpx structure. When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW , pututxline() invokes a program (that has the appropriate forced privileges) to verify and write the entry, since /etc/utmpx is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW . In this event, the <i>ut_name</i> field must correspond to the actual user name associated with the process; the <i>ut_type</i> field must be either USER_PROCESS or DEAD_PROCESS ; and the <i>ut_line</i> field must be a device special file and be writable by the user. If the process does not have the PAF_TRUSTED_PATH process attribute, all other fields in the entry are cleared.
setutxent()	Resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.
endutxent()	Closes the currently open file.
utmpxname()	Allows the user to change the name of the file examined, from /var/adm/utmpx to any other file. It is most often expected that this other file will be /var/adm/wtmpx . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. utmpxname() does not open the file. It just closes the old file if it is currently open and saves the new file name. The new file name must end with the “ x ” character to allow the name of the corresponding utmp file to be easily obtainable; otherwise, an error code of 1 is returned.
getutmp()	Copies the information stored in the fields of the utmpx structure to the corresponding fields of the utmp structure. If the information in any field of utmpx does not fit in the corresponding utmp field, the data is truncated. (See getutent(3C) for utmp structure)
getutmpx()	Copies the information stored in the fields of the utmp structure to the corresponding fields of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	Checks the existence of <i>wfile</i> and its parallel file, whose name is obtained by appending an “ x ” to <i>wfile</i> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <i>utmp</i> is written to <i>wfile</i> and the corresponding utmpx

structure is written to the parallel file.

updwtmpx() Checks the existence of *wfilex* and its parallel file, whose name is obtained by truncating the final “x” from *wfilex*. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. *utmpx* is written to *wfilex*, and the corresponding **utmp** structure is written to the parallel file.

RETURN VALUES A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

FILES

/var/adm/utmp	contains current user access and administrative information (old format)
/var/adm/utmpx	contains current user access and administration information (new format)
/var/adm/wtmp	contains a history of user access and administrative information.
/var/adm/wtmpx	contains a history of user access and administrative information.

SEE ALSO **getutent(3C)**, **ttyslot(3C)**, **utmp(4)**, **utmpx(4)**

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either **getutxid()** or **getutxline()**, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use **getutxline()** to search for multiple occurrences it would be necessary to zero out the static after each success, or **getutxline()** would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by **pututxline()** (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the **getutxent()**, **getutxid()**, or **getutxline()** routines, if the user has just modified those contents and passed the pointer back to **pututxline()**.

These routines use buffered standard I/O for input, but **pututxline()** uses an unbuffered write to avoid race conditions between processes trying to modify the **utmpx** and **wtmpx** files.

SUMMARY OF TRUSTED SOLARIS CHANGES

pututxline() invokes a program with appropriate forced privileges to verify and write the **utmpx** structure. **pututxline** clears fields in an entry if the process does not have the PAF_TRUSTED_PATH process attribute.

NAME	getvsaent, getvsafile – get vfstab_adjunct file entry
SYNOPSIS	<pre>#include <stdio.h> #include <tsol/vfstab_adjunct.h> int getvsaent(FILE *fp, struct vfsaent **vp); int getvsafile(FILE *fp, struct vfsaent **vp, char *file);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	getvsaent() and getvsafile() each fill in the structure pointed to by <i>vp</i> with the broken-out fields of a line in the <code>/etc/security/tsol/vfstab_adjunct</code> file. Each line in the file contains a vfstab_adjunct structure, declared in the <code><tsol/vfstab_adjunct.h></code> header: <pre>struct vfsaent { char *vsa_fsname; char *vsa_attrs; };</pre> The <i>vsa_fsname</i> contains the full pathname of the file system as listed in vfstab(4). The <i>vsa_attrs</i> points to the attribute string composed of keyword / value assignments of the form <i>keyword=value</i> separated by semicolons as described in vfstab_adjunct(4TSOL). getvsaent() returns a pointer to the next vfsaent structure in the file; so successive calls can be used to search the entire file. getvsafile() searches the file referred to by <i>fp</i> until a mount point matching <i>file</i> is found. On successful return, the locations referred to by <i>*vp</i>, <i>*vp->vsa_fsname</i>, and <i>*vp->vsa_attrs</i> have been separately allocated and may be independently released by calls to free(3). Note that these routines do not open, close, or rewind the file.
RETURN VALUES	If the next entry is successfully read by getvsaent() or a match is found with getvsafile(), 0 is returned. If an end-of-file is encountered on reading, these functions return -1. If an error is encountered, a value greater than 0 is returned. The possible error values are: ENOMEM Memory cannot be allocated for an entry.
NOTES	These interfaces are uncommitted, which means that even though they are not expected to change, they may change between minor releases of Trusted Solaris 2.x.
FILES	<code>/etc/security/tsol/vfstab_adjunct</code>
SEE ALSO	vfstab_adjunct (4TSOL)

NAME	hextob, htobcl, htobsl, htobil, htobclear – convert hexadecimal string to binary label
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> int htobcl(const char *s, bclabel_t *label) int htobsl(const char *s, bslabel_t *label) int htobil(const char *s, bilabel_t *label) int htobclear(const char *s, bclear_t *clearance)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	These functions convert hexadecimal string representations of internal label values into binary labels. htobcl() converts a hexadecimal string of the form: 0x<i>Information Label hexadecimal value</i> [0x<i>Sensitivity Label hexadecimal value</i>] into a binary CMW label. htobsl() converts a hexadecimal string of the form: 0x<i>Sensitivity Label hexadecimal value</i> into a binary sensitivity label. htobil() converts a hexadecimal string of the form: 0x<i>Information Label hexadecimal value</i> into a binary information label. htobclear() converts a hexadecimal string of the form: 0x<i>Clearance hexadecimal value</i> into a binary clearance.
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.
SEE ALSO	atohexlabel(1MTSOL), hextoalabel(1MTSOL), bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltcolor(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL), stobl(3TSOL) <i>Trusted Solaris Developer's Guide</i>

NAME	initgroups – initialize the supplementary group access list
SYNOPSIS	<pre>#include <grp.h> #include <sys/types.h> int initgroups(const char *name, gid_t basegid);</pre>
MT-LEVEL	Unsafe
DESCRIPTION	initgroups() reads the group database to get the group membership for the user specified by <i>name</i> and then initializes the supplementary group access list of the calling process (see getgrnam(3C) and getgroups(2TSOL)). The <i>basegid</i> group id is also included in the supplementary group access list. This is typically the real group id from the user database. While scanning the group database, if the number of groups, including the <i>basegid</i> entry, exceeds {NGROUPS_MAX}, subsequent group entries are ignored.
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and errno is set to indicate the error.
ERRORS	initgroups() will fail and not change the supplementary group access list if: EPERM The effective user id is not superuser.
SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, initgroups() must have PRIV_PROC_SETID in its set of effective privileges.
SEE ALSO	getgroups (2TSOL), getgrnam (3C)

NAME	kstat_read, kstat_write – read or write kstat data
SYNOPSIS	<pre>cc [flag ...] file ... -lkstat [library...] #include <kstat.h> kid_t kstat_read(kstat_ctl_t *kc, kstat_t *ksp, void *buf); kid_t kstat_write(kstat_ctl_t *kc, kstat_t *ksp, void *buf);</pre>
DESCRIPTION	kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp->ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp->ks_ndata</i> is set to the number of data fields, <i>ksp->ks_data_size</i> is set to the total size of the data, and <i>ksp->ks_snaptime</i> is set to the high-resolution time at which the data snapshot was taken. If <i>buf</i> is non-NULL, the data is copied from <i>ksp->ks_data</i> into <i>buf</i>. kstat_write() writes data from <i>buf</i>, or from <i>ksp->ks_data</i> if <i>buf</i> is NULL, to the corresponding kstat in the kernel. kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.
RETURN VALUES	On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return -1.
FILES	/dev/kstat kernel statistics driver
SUMMARY OF TRUSTED SOLARIS CHANGES	kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat .
SEE ALSO	kstat(3K) , kstat_chain_update(3K) , kstat_close(3K) , kstat_data_lookup(3K) , kstat_lookup(3K) , kstat_open(3K)

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ltsol -lDtSol [<i>library</i> ...]</pre> <pre>#include <Dt/ModLabel.h></pre> <pre>ModLabelData *tsol_lbuild_create(Widget widget, void (*event_handler)() ok_callback, lbuild_attributes extended_operation, ... , NULL)</pre> <pre>void *tsol_lbuild_get(ModLabelData *data, lbuild_attributes extended_operation)</pre> <pre>void tsol_lbuild_set(ModLabelData *data, lbuild_attributes extended_operation, ... , NULL)</pre> <pre>void tsol_lbuild_destroy(ModLabelData *data)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the label_encodings(4TSOL) file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options. Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the tsol_lbuild_create() call where it is mapped to the pushbutton widget. When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the PRIV_SYS_TRANS_LABEL privilege in its effective set. If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are greyed. There are no privileges to override these restrictions. tsol_lbuild_create() creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the tsol_lbuild_create() input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the tsol_lbuild_get() and tsol_lbuild_set() routines. The widget information is accessed directly by referencing the following fields of the

ModLabeldata structure.

<i>lbuild_dialog</i>	The label builder dialog box.
<i>ok</i>	The Ok pushbutton.
<i>cancel</i>	The Cancel pushbutton.
<i>reset</i>	The Reset pushbutton.
<i>help</i>	The Help pushbutton.

The **tsol_lbuild_create()** parameter list takes the following values:

<i>widget</i>	The widget from which the dialog box is created. Any Motif widget can be passed.
<i>ok_callback</i>	A callback function that implements the behavior of the OK pushbutton on the dialog box.
<i>..., NULL</i>	A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

tsol_lbuild_destroy() destroys the ModLabelData structure returned by **tsol_lbuild_create()**.

tsol_lbuild_get() and **tsol_lbuild_set()** access the information stored in the ModLabel-Data structure returned by **tsol_lbuild_create()**.

The following extended operations can be passed to **tsol_lbuild_create()** to build the user interface, to **tsol_lbuild_get()** to retrieve information on the user interface, and to **tsol_lbuild_set()** to change the user interface information. All extended operations are valid for **tsol_lbuild_get()**, but the *WORK* operations are not valid for **tsol_lbuild_set()** or **tsol_lbuild_create()** because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE	Create a user interface to build an informaiton label, sensitivity label, CMW label, or clearance. Value is LBUILD_MODE_CMW by default.
	LBUILD_MODE_IL - Build an information label.
	LBUILD_MODE_SL - Build a sensitivity label
	LBUILD_MODE_CMW - Build a CMW label
	LBUILD_MODE_CLR - Build a clearance
LBUILD_VALUE_SL	The starting sensitivity label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_SL .
LBUILD_VALUE_IL	The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW	The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW.
LBUILD_VALUE_CLR	The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR.
LBUILD_USERFIELD	A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.
LBUILD_SHOW	Show or hide the Label builder dialog box. Value is FALSE by default. TRUE - Show the Label builder dialog box. FALSE - Hide the Label builder dialog box.
LBUILD_TITLE	A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.
LBUILD_WORK_SL	Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.
LBUILD_WORK_IL	Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.
LBUILD_WORK_CMW	Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.
LBUILD_WORK_CLR	Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The clearance the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.
LBUILD_X	The X position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.
LBUILD_Y	The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.
LBUILD_LOWER_BOUND	The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND	The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A value you supply should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the PRIV_SYS_TRANS_LABEL privilege, the value is the user's clearance.
LBUILD_CHECK_AR	Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.
LBUILD_VIEW	Use the internal or external label representation. Value is LBUILD_VIEW_EXTERNAL by default. LBUILD_VIEW_INTERNAL - Use the internal names for the highest and lowest labels in the system: ADMIN_HIGH and ADMIN_LOW. LBUILD_VIEW_EXTERNAL - Promote an ADMIN_LOW label to the next highest label, and demote an ADMIN_HIGH label to the next lowest label.

RETURN VALUES

The **tsol_lbuild_get()** returns -1 if it is unable to get the value.

The **tsol_lbuild_create()** routine returns a variable of type **ModLabelData** that contains the information provided in the **tsol_lbuild_create()** input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES

This example calls the **tsol_lbuild_create()** routine.

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
                                         LBUILD_MODE,          LBUILD_MODE_CMW,
                                         LBUILD_TITLE,          "Setting CMW Label",
                                         LBUILD_VIEW,            LBUILD_VIEW_INTERNAL
                                         LBUILD_X,                200,
                                         LBUILD_Y,                200,
                                         LBUILD_USERFIELD,        "Pathname:",
                                         LBUILD_SHOW,             FALSE,
                                         NULL);
```

These examples call the **tsol_lbuild_get()** routine to query the mode being used, and calls the **tsol_lbuild_set()** routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );
tsol_lbuild_set(lbldata,
               LBUILD_SHOW,    TRUE,
               NULL );
```

This example destroys the ModLabelData variable returned in the call to **tsol_lbuild_create()**.

```
tsol_lbuild_destroy(lbldata);
```

FILES **/usr/dt/include/Dt/ModLabel.h**
 /etc/security/tsol/label_encodings

SEE ALSO **label_encodings(4TSOL)**
 Trusted Solaris Developer's Guide
 Trusted Solaris administrator's document set
 Trusted Solaris Label Administration

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ltsol -lDtTsol [<i>library</i> ...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString Xbcltos(Display *<i>display</i>, const bclabel_t *<i>cmwlabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString Xbsltos(Display *<i>display</i>, const bxlabel_t *<i>senslabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString Xbiltos(Display *<i>display</i>, const bilabel_t *<i>inflabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString Xbcleartos(Display *<i>display</i>, const bclear_t *<i>clearance</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to translate labels or clearances that dominate the current process's sensitivity label. <i>display</i> The structure controlling the connection to an X Window System display. <i>cmwlabel</i> The CMW label to be translated. <i>senslabel</i> The sensitivity label to be translated. <i>inflabel</i> The information label to be translated. <i>clearance</i> The clearance to be translated. <i>width</i> The width of the translated label or clearance in pixels. If the specified width is shorter than the full label, the label is clipped and the presence of clipped letters is indicated by an arrow. In this example, letters have been clipped to the right of: TS<-. See the sbltos(3TSOL) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word wrapped using a width of half the display width. <i>fontlist</i> A list of fonts and character sets where each font is associated with a character set. <i>flags</i> The value of flags indicates which words in the label_encodings(4TSOL) file are used for the translation. See the bltos(3TSOL) man page for a description of the flag values: LONG_WORDS, SHORT_WORDS, LONG_CLASSIFICATION, SHORT_CLASSIFICATION, ALL_ENTRIES, ACCESS_RELATED,

VIEW_EXTERNAL, VIEW_INTERNAL, NO_CLASSIFICATION. BRACKETED is an additional flag that can be used with **Xbsltos()** only. It encloses the sensitivity label in square brackets as follows: [C].

RETURN VALUES

These interfaces return a compound string that represents the ASCII form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in *fontlist* and is clipped to *width*.

These interfaces return NULL if the label or clearance is not a valid, required type as defined in the **label_encodings**(4TSOL) file, or not dominated by the process's sensitivity label and the **PRIV_SYS_TRANS_LABEL** privilege is not asserted.

EXAMPLES

This example translates a clearance to ASCII using the long words specified in the **label_encodings**(4TSOL) file, a font list, and clips the translated clearance to a width of 72 pixels.

```
xmstr = Xbcleartos(XtDisplay(topLevel), &clearance, 72, fontlist, LONG_WORDS );
```

FILES

/usr/dt/include/Dt/label_clipping.h
/etc/security/tsol/label_encodings

SEE ALSO

label_encodings(4TSOL), **bltos**(3TSOL), **sbltos**(3TSOL), **XmStringDraw**(3X), and **FontList**(3X) for information on the creation and structure of a font list.

Trusted Solaris Developer's Guide

Trusted Solaris administrator's document set

Trusted Solaris Label Administration

NAME	labelinfo – get information about the label encodings																		
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> struct label_info { short ilabel_len; /* max Information Label length */ short slabel_len; /* max Sensitivity Label length */ short clabel_len; /* max CMW Label length */ short clear_len; /* max Clearance Label length */ short vers_len; /* version string length */ short header_len; /* max len of banner page header */ short protect_as_len; /* max len of banner page protect as */ short caveats_len; /* max len of banner page caveats */ short channels_len; /* max len of banner page channels */ }; int labelinfo(struct label_info *info)</pre>																		
AVAILABILITY	SUNWtsolu																		
MT-LEVEL	MT-Safe																		
DESCRIPTION	Information about the label encodings is returned in the label_info structure pointed to by <i>info</i>. The fields in this structure have the following values: <table> <tr> <td><i>ilabel_len</i></td><td>The maximum length of an <i>character-coded Information Label</i> returned when translated from a <i>Binary Information Label</i>."</td></tr> <tr> <td><i>slabel_len</i></td><td>The maximum length of an <i>character-coded Sensitivity Label</i> returned when translated from a <i>Binary Sensitivity Label</i> .</td></tr> <tr> <td><i>clabel_len</i></td><td>The maximum length of an <i>character-coded CMW Label</i> returned when translated from a <i>Binary CMW Label</i>.</td></tr> <tr> <td><i>clear_len</i></td><td>The maximum length of an <i>character-coded Clearance</i> returned when translated from a <i>Binary Clearance</i>.</td></tr> <tr> <td><i>vers_len</i></td><td>The length of the label_encodings file version string returned by labelvers().</td></tr> <tr> <td><i>header_len</i></td><td>The maximum length of a printer banner page <i>header</i> string returned by bcltobanner().</td></tr> <tr> <td><i>protect_as_len</i></td><td>The maximum length of a printer banner page <i>protect_as</i> string returned by bcltobanner().</td></tr> <tr> <td><i>caveats_len</i></td><td>The maximum length of a printer banner page <i>caveats</i> string returned by bcltobanner().</td></tr> <tr> <td><i>channels_len</i></td><td>The maximum length of a printer banner page <i>channels</i> string returned by bcltobanner().</td></tr> </table>	<i>ilabel_len</i>	The maximum length of an <i>character-coded Information Label</i> returned when translated from a <i>Binary Information Label</i> ."	<i>slabel_len</i>	The maximum length of an <i>character-coded Sensitivity Label</i> returned when translated from a <i>Binary Sensitivity Label</i> .	<i>clabel_len</i>	The maximum length of an <i>character-coded CMW Label</i> returned when translated from a <i>Binary CMW Label</i> .	<i>clear_len</i>	The maximum length of an <i>character-coded Clearance</i> returned when translated from a <i>Binary Clearance</i> .	<i>vers_len</i>	The length of the label_encodings file version string returned by labelvers() .	<i>header_len</i>	The maximum length of a printer banner page <i>header</i> string returned by bcltobanner() .	<i>protect_as_len</i>	The maximum length of a printer banner page <i>protect_as</i> string returned by bcltobanner() .	<i>caveats_len</i>	The maximum length of a printer banner page <i>caveats</i> string returned by bcltobanner() .	<i>channels_len</i>	The maximum length of a printer banner page <i>channels</i> string returned by bcltobanner() .
<i>ilabel_len</i>	The maximum length of an <i>character-coded Information Label</i> returned when translated from a <i>Binary Information Label</i> ."																		
<i>slabel_len</i>	The maximum length of an <i>character-coded Sensitivity Label</i> returned when translated from a <i>Binary Sensitivity Label</i> .																		
<i>clabel_len</i>	The maximum length of an <i>character-coded CMW Label</i> returned when translated from a <i>Binary CMW Label</i> .																		
<i>clear_len</i>	The maximum length of an <i>character-coded Clearance</i> returned when translated from a <i>Binary Clearance</i> .																		
<i>vers_len</i>	The length of the label_encodings file version string returned by labelvers() .																		
<i>header_len</i>	The maximum length of a printer banner page <i>header</i> string returned by bcltobanner() .																		
<i>protect_as_len</i>	The maximum length of a printer banner page <i>protect_as</i> string returned by bcltobanner() .																		
<i>caveats_len</i>	The maximum length of a printer banner page <i>caveats</i> string returned by bcltobanner() .																		
<i>channels_len</i>	The maximum length of a printer banner page <i>channels</i> string returned by bcltobanner() .																		

RETURN VALUES	labelinfo() returns 1 on success and -1 if the label_encodings information is unavailable.
FILES	/etc/security/tsol/label_encodings The label encodings file containing the <i>Classification</i> names, <i>Words</i> , constraints, and values for the defined labels of this system.
SEE ALSO	bcltobanner(3TSOL)C , bilconjoin(3TSOL)C , blcompare(3TSOL)C , blinset(3TSOL)C , blmanifest(3TSOL)C , blminmax(3TSOL)C , blportion(3TSOL)C , bltcolor(3TSOL)C , bltos(3TSOL)C , bltype(3TSOL)C , blvalid(3TSOL)C , btohex(3TSOL)C , hextob(3TSOL)C , labelvers(3TSOL)C , sbltos(3TSOL)C , stobl(3TSOL)C , label_encodings(4TSOL) <i>Trusted Solaris Developer's Guide</i>
WARNINGS	If the label_encodings file is modified after an application gets information about it, that information may be out of date.
BUGS	The label_encodings file is rarely updated on a running system and there is no way of informing an application that the label_encodings file has been modified.

NAME	labelvers – get version of the label_encodings file
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> int labelvers(char **version, const int len)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	<i>version</i> may point either to a pointer to pre-allocated memory or to the value (char *)0. If <i>version</i> points to a pointer to pre-allocated memory, then <i>len</i> indicates the size of that memory. If <i>version</i> points to the value (char *)0, memory is allocated using malloc() to contain the label_encodings file version string. The version string from the label_encodings file is copied into the allocated or pre-allocated memory.
RETURN VALUES	labelvers() returns: -1 If the label_encodings file is inaccessible. 0 If memory cannot be allocated for the return string or if the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (* <i>version</i>[0] = '\000'). > 0 If successful, the length of the version string including the NULL terminator.
FILES	/etc/security/tsol/label_encodings The label encodings file containing the <i>Classification</i> names, <i>Words</i>, constraints, and values for the defined labels of this system.
SEE ALSO	bcltobanner(3TSOL), bilconjoin(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltcolor(3TSOL), bltos(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), free(3C), hextob(3TSOL), labelinfo(3TSOL), malloc(3C), sbltos(3TSOL), stobl(3TSOL), label_encodings(4TSOL) <i>Trusted Solaris Developer's Guide</i>
WARNINGS	If the label_encodings file is modified after the version string is obtained, that string may be out of date.
NOTES	If memory is allocated by this routine, the caller must free memory with free() when the memory is no longer in use.

BUGS

The **label_encodings** file is rarely updated on a running system, and there is no way of informing an application that the **label_encodings** file has been modified.

NAME	libt6 – TSIX trusted IPC library
SYNOPSIS	#include <tsix/t6attrs.h>
AVAILABILITY	Available on Trusted Solaris systems and on all other TSIX(RE) 1.1–API compliant systems
MT-LEVEL	MT-Safe
DESCRIPTION	libt6 constitutes the TSIX Application Program Interface (API). It is a library of routines that an application uses to control attribute transport during trusted interprocess communication. The routines in the library are recommended over the underlying system call interfaces for portability because they shield the application from operating system, communication protocol, and IPC mechanism specifics. The libt6 routines provide interfaces through which the trusted application: • Specifies the security attributes used to label outgoing IPC messages (<i>on-message attributes</i>) and reads the on-message attributes associated with a received message. • Controls the security options of the endpoint used to perform trusted IPC.
SECURITY ATTRIBUTES	The security attributes associated with the sending process are called on-message attributes because they are independent of the contents of the message. The TCBs decide what to do with the message based on the on-message attributes. The security attributes associated with a process, and therefore those that are used to label IPC messages, vary with the configuration of the system but must be a subset of the following attributes: - Sensitivity label - Nationality Caveats - Integrity Label - TSIX Session ID - Clearance - Access Control List - Information label - Effective Privileges - Audit ID - Process ID - Additional Audit Information - Process Attributes - User ID - Group ID - Supplementary Group IDs NOTE: Some of these attributes imply component security policies that may not be available on some systems. The TSIX application program interface allows trusted applications to change the on-message attributes associated with an outgoing message and retrieve the on-message attributes associated with an incoming message.

**ON-MESSAGE
ATTRIBUTE
ROUTINES**

The on-message attribute routines affect the security attributes associated with outgoing messages or retrieve attributes associated with incoming messages. The caller specifies attributes to these routines through a **t6attr_t** control structure (defined in **<tsix/t6attr.h>**, an opaque structure used to access sets of security attributes. The caller specifies the attributes applied to outbound messages or retrieved from incoming messages through TSIX routines. Specified attributes are copied from or written to the buffers accessible through the control structure. Any attributes not designated by the sender are supplied for outgoing messages by the underlying trusted kernel. The routines that send and retrieve on-message attributes operate on sockets or streams, generically referred to as endpoints.

- t6alloc_blk(3NTSOL)C** Allocates space for a **t6attr_t** control structure.
- t6free_blk(3NTSOL)** Frees attribute control structure and buffers. This interface should be used in conjunction with **t6alloc_blk(3NTSOL)C**, which allocates the space.
- t6dup_blk(3NTSOL)** Given one attribute control structure, this routine allocates enough storage to hold a duplicate control structure and all attributes it references, and creates a duplicate.
- t6copy_blk(3NTSOL)** Copies a **t6attr_t** control structure and the security attributes to which it points into a second, previously allocated **t6attr_t** structure and its previously allocated buffers.
- t6clear_blk(3NTSOL)** Clears attributes from a **t6attr_t** control structure.
- t6cmp_blk(3NTSOL)** Compares two **t6attr_t** control structures for equal attributes set.
- t6size_attr(3NTSOL)** Gets the size of an attribute from the control structure.
- t6get_attr(3NTSOL)** Gets an attribute handled by the control structure.
- t6set_attr(3NTSOL)** Sets an attribute handled by the control structure.
- t6sendto(3NTSOL)** Sends data and a specified set of security attributes on a endpoint.
- t6recvfrom(3NTSOL)** Reads a network message and retrieves the security attributes associated with the data.
- t6peek_attr(3NTSOL)** Peeks ahead and returns the attributes associated with the next byte of data.
- t6last_attr(3NTSOL)** Returns the security attributes associated with the last byte of data read from the network endpoint.
- t6get_endpt_mask(3NTSOL)**
Gets the endpoint mask.
- t6set_endpt_mask(3NTSOL)**
Sets the endpoint mask.
- t6get_endpt_default(3NTSOL)**
Gets the endpoint default security attributes.
- t6set_endpt_default(3NTSOL)**

NETWORK ENDPOINT SECURITY OPTIONS	Sets the endpoint default security attributes. t6attr_query(3NTSOL) Gets a mask that indicates which attributes came from templates. A trusted application can manipulate a number of security options associated with the network endpoint via the following calls: t6ext_attr(3NTSOL) Turns on or off the security extensions to the network endpoint. This must be called before using any other libt6 routines. t6new_attr(3NTSOL) Specifies to the network endpoint that the receiving process is only interested in receiving attributes if they have changed since the last time it received them. This saves the overhead created by passing attributes unnecessarily with each message.
INCLUDE FILES	Any programs that use routines in this library must include the header files containing declarations pertinent to the routine. The synopsis section of each manual page indicates the required header files. Most routines in the library contain references to declarations defined in <tsix/t6attrs.h>. This file defines constants for attribute types to be used by various TSIX attribute library access functions, as well as constants used as parameters to the library functions.
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document.
TRUSTED SOLARIS SECURITY ATTRIBUTES	Trusted Solaris 2 supports the following security attributes: - Sensitivity Label - Clearance - Information Label - Effective Privileges - Process Attributes - Effective User ID - Effective Group ID Trusted Solaris 2.5 also supports the following attributes as read only: - Session ID - Access Control List - Audit ID - Process ID - Additional Audit Information - Supplemental Group IDs

NAME	listen – listen for connections on a socket						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...]</pre> <pre>#include <sys/types.h></pre> <pre>#include <sys/socket.h></pre> <pre>int listen(int <i>s</i>, int <i>backlog</i>);</pre>						
MT-LEVEL	Safe						
DESCRIPTION	To accept connections, a socket is first created with socket(3N), a backlog for incoming connections is specified with listen() and then the connections are accepted with accept(3NTSOL). The listen() call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET. The <i>backlog</i> parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full, the client will receive an error with an indication of ECONNREFUSED for AF_UNIX sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For AF_INET sockets, the tcp will retry the connection. If the <i>backlog</i> is not cleared by the time the tcp times out, the connect will fail with ETIMEDOUT.						
SUMMARY OF TRUSTED SOLARIS CHANGES	If the calling process possesses the PRIV_NET_MAC_READ privilege, the endpoint will bind to an MLP; otherwise, it will bind to a SLP.						
RETURN VALUES	A 0 return value indicates success; -1 indicates an error.						
ERRORS	The call fails if: <table> <tr> <td>EBADF</td><td>The argument <i>s</i> is not a valid file descriptor.</td></tr> <tr> <td>ENOTSOCK</td><td>The argument <i>s</i> is not a socket.</td></tr> <tr> <td>EOPNOTSUPP</td><td>The socket is not of a type that supports the operation listen().</td></tr> </table>	EBADF	The argument <i>s</i> is not a valid file descriptor.	ENOTSOCK	The argument <i>s</i> is not a socket.	EOPNOTSUPP	The socket is not of a type that supports the operation listen() .
EBADF	The argument <i>s</i> is not a valid file descriptor.						
ENOTSOCK	The argument <i>s</i> is not a socket.						
EOPNOTSUPP	The socket is not of a type that supports the operation listen() .						
SEE ALSO	accept(3NTSOL) , connect(3N) , socket(3N)						
NOTES	There is currently no <i>backlog</i> limit.						

NAME	mldgetcwd – get pathname of current working directory
SYNOPSIS	<pre>#include <unistd.h> #include <tsol/mld.h> extern char *mldgetcwd(char *buf, size_t size);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	mldgetcwd() returns a pointer to the current directory pathname including any MLD adornments and SLD names. The value of <i>size</i> must be at least one greater than the length of the pathname to be returned. If <i>buf</i> is not NULL, the pathname will be stored in the space pointed to by <i>buf</i>. If <i>buf</i> is a NULL pointer, mldgetcwd() will obtain <i>size</i> bytes of space using malloc(3C). In this case, the pointer returned by mldgetcwd() may be used as the argument in a subsequent call to free().
RETURN VALUES	mldgetcwd() returns NULL with errno set if <i>size</i> is not large enough, or if an error occurs in a lower-level function.
ERRORS	mldgetcwd() will fail if one or more of the following are true: EACCES A parent directory cannot be read to get its name. EINVAL <i>size</i> is equal to 0. ERANGE <i>size</i> is greater than 0 and less than the length of the pathname plus 1.
EXAMPLE	Here is a program that prints the current working directory. <pre>#include <unistd.h> #include <stdio.h> main() { char *cwd; if ((cwd = mldgetcwd(NULL, 64)) == NULL) { perror("pwd"); exit(2); } (void)printf("%s\n", cwd); return(0); }</pre>

SEE ALSO**chdir**(2TSOL), **malloc**(3C)**NOTES**Using **chdir**(2TSOL) in conjunction with **mldgetcwd** can give unpredictable results.

NAME	mldrealpath, mldrealpathl – return the canonicalized absolute pathname, including any MLD adornments and SLD names.	
SYNOPSIS	<pre>#include <sys/param.h> #include <tsol/mld.h> char *mldrealpath(const char *path, char *resolved_path) #include <tsol/label.h> char *mldrealpathl(const char *path, char *resolved_path, const bslabel_t *sl)</pre>	
AVAILABILITY	SUNWtsolu	
MT-LEVEL	MT-Safe	
DESCRIPTION	mldrealpath() expands all symbolic links and resolves references to <code>'/./'</code>, <code>'/..'</code>, extra <code>'/'</code> characters, and MLD translations in the NULL terminated string named by <i>path</i> and stores the canonicalized absolute pathname in the buffer named by <i>resolved_path</i>. The resulting path will have no symbolic links components, nor any <code>'/./'</code>, <code>'/..'</code>, nor any unadorned MLDs, nor any hidden SLD names for single level directories at the present Sensitivity Label. mldrealpathl() operates the same as mldrealpath() except the SLD name is relative to the Sensitivity Label <i>sl</i>. To specify a Sensitivity Label for an SLD name which doesn't exist, the process must assert either the PRIV_FILE_UPGRADE_SL or PRIV_FILE_DOWNGRADE_SL privilege depending on whether the specified Sensitivity Label dominates or does not dominate the process Sensitivity Label.	
RETURN VALUES	mldrealpath() returns a pointer to the <i>resolved_path</i> on success. On failure, it returns NULL , sets errno to indicate the error, and places in <i>resolved_path</i> the absolute pathname of the <i>path</i> component which could not be resolved.	
ERRORS	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> .
	EFAULT	<i>resolved_path</i> extends outside the process's allocated address space.
	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
	EINVAL	<i>path</i> or <i>resolved_path</i> was NULL .
	EIO	An I/O error occurred while reading from or writing to the file system.
	ENAMETOOLONG	The length of the path argument exceeds {PATH_MAX}. A pathname component is longer than {NAME_MAX} (see sysconf(2V)) while {_POSIX_NO_TRUNC} is in effect (see pathconf(2V)).

ENOENT The named file does not exist.

SEE ALSO **readlink(2TSOL)**, **getsldname(2TSOL)**, **mldgetwd(3TSOL)**

WARNINGS It indirectly invokes the **readlink(2TSOL)** system call and **mldgetwd(3TSOL)** library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

NAME	mlock, munlock – lock (or unlock) pages in memory	
SYNOPSIS	<pre>#include <sys/types.h> int mlock(caddr_t addr, size_t len); int munlock(caddr_t addr, size_t len);</pre>	
MT-LEVEL	MT-Safe	
DESCRIPTION	The function mlock() uses the mappings established for the address range <i>[addr, addr + len)</i> to identify pages to be locked in memory. If the page identified by a mapping changes, such as occurs when a copy of a writable MAP_PRIVATE page is made upon the first store, the lock will be transferred to the newly copied private page. munlock() removes locks established with mlock(). A given page may be locked multiple times by executing an mlock() through different mappings. That is, if two different processes lock the same page, then the page will remain locked until both processes remove their locks. However, within a given mapping, page locks do not nest – multiple mlock() operations on the same address in the same process will all be removed with a single munlock(). Of course, a page locked in one process and mapped in another (or visible through a different mapping in the locking process) is still locked in memory. This fact can be used to create applications that do nothing other than lock important data in memory, thereby avoiding page I/O faults on references from other processes in the system. If the mapping through which an mlock() has been performed is removed, an munlock() is implicitly performed. An munlock() is also performed implicitly when a page is deleted through file removal or truncation. Locks established with mlock() are not inherited by a child process after a fork() and are not nested. Because of the impact on system resources, the use of mlock() and munlock() is restricted to users in administrative roles with the PRIV_SYS_CONFIG privilege. Attempts to mlock() more memory than a system-specific limit will fail.	
RETURN VALUES	Upon successful completion, the functions mlock() and munlock() return 0; otherwise, they return –1 and set errno to indicate the error.	
ERRORS	EAGAIN	mlock() only. Some or all of the memory identified by the range <i>[addr, addr + len)</i> could not be locked because of insufficient system resources.
	EINVAL	<i>addr</i> is not a multiple of the page size as returned by sysconf(3C) .
	ENOMEM	Addresses in the range <i>[addr, addr + len)</i> are invalid for the address space of a process, or specify one or more pages which are not mapped.
	EPERM	The process does not have sufficient privilege.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

To succeed, **mlock()** must have PRIV_SYS_CONFIG in its set of effective privileges

SEE ALSO

fork(2TSOL), **memcntl(2)**, **mmap(2)**, **plock(3CTSOL)**, **mlockall(3CTSOL)**, **sysconf(3C)**

NOTES

mlock and **munlock** require the **PRIV_SYS_CONFIG** privilege.

NAME	mlockall, munlockall – lock or unlock address space						
SYNOPSIS	<pre>#include <sys/mman.h> int mlockall(int flags); int munlockall(void);</pre>						
MT-LEVEL	MT-Safe						
DESCRIPTION	The function mlockall() causes all pages mapped by an address space to be locked in memory. The value of <i>flags</i> determines whether the pages to be locked are those currently mapped by the address space, those that will be mapped in the future, or both: <table> <tr> <td>MCL_CURRENT</td><td>Lock current mappings</td></tr> <tr> <td>MCL_FUTURE</td><td>Lock future mappings</td></tr> </table> If MCL_FUTURE is specified to mlockall(), then as mappings are added to the address space (or existing mappings are replaced) they will also be locked, provided sufficient memory is available. Mappings locked via mlockall() with any option may be explicitly unlocked with a munlock() call. The function munlockall() removes address space locks and locks on mappings in the address space. All conditions and constraints on the use of locked memory as exist for mlock() apply to mlockall(). Locks established with mlockall() are not inherited by a child process after a fork() and are not nested.	MCL_CURRENT	Lock current mappings	MCL_FUTURE	Lock future mappings		
MCL_CURRENT	Lock current mappings						
MCL_FUTURE	Lock future mappings						
RETURN VALUES	Upon successful completion, the functions mlockall() and munlockall() return 0; otherwise, they return –1 and set errno to indicate the error.						
ERRORS	<table> <tr> <td>EAGAIN</td><td>mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.</td></tr> <tr> <td>EINVAL</td><td><i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE.</td></tr> <tr> <td>EPERM</td><td>The process does not have sufficient privilege.</td></tr> </table>	EAGAIN	mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.	EINVAL	<i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE .	EPERM	The process does not have sufficient privilege.
EAGAIN	mlockall() only. Some or all of the memory in the address space could not be locked due to sufficient resources.						
EINVAL	<i>flags</i> contains values other than MCL_CURRENT and MCL_FUTURE .						
EPERM	The process does not have sufficient privilege.						
SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, mlockall() must have PRIV_SYS_CONFIG in its set of effective privileges.						
SEE ALSO	fork(2TSOL) , memcntl(2) , mmap(2) , plock(3CTSOL) , mlock(3CTSOL) , sysconf(3C)						

NOTES

mlockall() and **munlockall()** require the **PRIV_SYS_CONFIG** privilege.

NAME	<code>nis_groups</code> , <code>nis_ismember</code> , <code>nis_addmember</code> , <code>nis_removemember</code> , <code>nis_creategroup</code> , <code>nis_destroygroup</code> , <code>nis_verifygroup</code> , <code>nis_print_group_entry</code> , <code>nis_map_group</code> , <code>__nis_map_group</code> – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(const nis_name principal, const nis_name group); nis_error nis_addmember(const nis_name member, const nis_name group); nis_error nis_removemember(const nis_name member, const nis_name group); nis_error nis_creategroup(const nis_name group, const u_long flags); nis_error nis_destroygroup(const nis_name group); void nis_print_group_entry(const nis_name group); nis_error nis_verifygroup(const nis_name group);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object. The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object. There are three types of group members: • An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." • An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. • A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members. A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members. The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE. The <code>nis_addmember()</code> and <code>nis_removemember()</code> functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, <code>nis_addmember()</code> and

nis_removemember() must inherit the PAF_TRUSTED_PATH attribute.

The **nis_creategroup()** and **nis_destroygroup()** functions create and destroy group objects. The user must have create or destroy rights, respectively, for the *groups_dir* directory in the appropriate domain. The parameter *flags* to **nis_creategroup()** is currently unused and should be set to zero. To succeed, **nis_creategroup()** and **nis_destroygroup()** must inherit the PAF_TRUSTED_PATH attribute.

The **nis_print_group_entry()** function lists a group's members on the standard output.

The **nis_verifygroup()** function returns **NIS_SUCCESS** if the given group exists, otherwise it returns an error code.

EXAMPLES

Simple Memberships

Given a group **sadsouls.oz.** with members **tinman.oz.**, **lion.oz.**, and **scarecrow.oz.**, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

Implicit Memberships

Given a group **baddies.oz.**, with members **wickedwitch.west.oz.** and ***.monkeys.west.oz.**, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the **monkeys.west.oz.** domain belongs to the implicit group ***.monkeys.west.oz.**, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

Recursive Memberships

Given a group **goodandbad.oz.**, with members **toto.kansas**, **@sadsouls.oz.**, and **@baddies.oz.**, and the groups **sadsouls.oz.** and **baddies.oz.** defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because **wickedwitch.west.oz.** is a member of the **baddies.oz.** group which is recursively included in the **goodandbad.oz.** group.

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, **nis_addmember()**, **nis_removemember()**, **nis_creategroup()**, and **nis_destroygroup()** must inherit the PAF_TRUSTED_PATH attribute.

SEE ALSO

nisgrpadm(1), **nis_objects(3N)**

NOTES

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see **nis_objects(3N)**) with a variant part that is defined in the **group_obj** structure. It contains the following fields:

```
    u_long      gr_flags;           /* Interpretation Flags
                                     (currently unused) */

    struct {
        u_int     gr_members_len;
        nis_name  *gr_members_val;
    } gr_members;                   /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the **nis_servstate()** function with tag **TAG_GCACHE** and a value of 1.

There are currently no known methods for **nis_ismember()**, **nis_print_group_entry()**, and **nis_verifygroup()** to get their answers from only the master server.

NAME	<code>nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult</code> – NIS+ namespace functions
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_lookup(const nis_name name, const u_long flags); nis_result *nis_add(const nis_name name, const nis_object *obj); nis_result *nis_remove(const nis_name name, const nis_object *obj); nis_result *nis_modify(const nis_name name, const nis_object *obj); void nis_freeresult(nis_result *result);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions are used to locate and manipulate all NIS+ objects (see nis_objects(3N)) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to nis_subr(3N). nis_lookup() resolves a NIS+ name and returns a copy of that object from a NIS+ server. nis_add() and nis_remove() add and remove objects to the NIS+ namespace, respectively. nis_modify() can change specific attributes of an object that already exists in the namespace. These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in nis_subr(3N) should be used. nis_freeresult() frees all memory associated with a nis_result structure. This function must be called to free the memory associated with a NIS+ result. nis_lookup(), nis_add(), nis_remove(), and nis_modify() all return a pointer to a nis_result structure which <i>must</i> be freed by calling nis_freeresult() when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with nis_clone_object(3N) (see nis_subr(3N)). To succeed, nis_add(), nis_modify(), and nis_remove() must inherit the PAF_TRUSTED_PATH attribute. nis_lookup() takes two parameters, the name of the object to be resolved in <i>name</i>, and a flags parameter, <i>flags</i>, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see nis_tables(3N)). The nis_lookup() function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag EXPAND_NAME <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error NIS_BADNAME. The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list. FOLLOW_LINKS When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the

	linked object is itself a link, then this process will iterate until the either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object's domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function nis_getnames() (see nis_subr(3N)) which uses the environment variable NIS_PATH .

The status value may be translated to ascii text using the function **nis_sperrno()** (see **nis_error(3N)**).

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function **nis_add()** will take the object *obj* and add it to the NIS+ namespace with the name *name*. This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function **nis_remove()** will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not NULL, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the NIS_NOTSAMEOBJ error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function **nis_modify()** will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error NIS_NOTSAMEOBJ if

the object identifier of the passed object does not match that of the object being modified in the namespace.

Note: Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-NULL the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.'(dot) characters. If these conditions are not met the operation will fail and return the NIS_BADNAME error code.

Results

These functions return a pointer to a structure of type **nis_result**:

```
struct nis_result {
    nis_error status;
    struct {
        u_int objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    u_long      zticks;
    u_long      dticks;
    u_long      aticks;
    u_long      cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function **nis_sperrno()** (see **nis_error(3N)**).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to **nis_freeresult()**. If you need to keep a copy of one or more objects, they can be copied with the function **nis_clone_object()** and freed with the function **nis_destroy_object()** (see **nis_server(3N)**). Refer to **nis_objects(3N)** for a description of the **nis_object** structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see **nis_db(3N)**).

<i>zticks</i>	The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
<i>dticks</i>	The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
<i>aticks</i>	The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
<i>cticks</i>	The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the

sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_SUCCESS	The request was successful.
NIS_S_SUCCESS	The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag NO_CACHE when you call the lookup function.
NIS_NOTFOUND	The named object does not exist in the namespace.
NIS_CACHEEXPIRED	The object returned came from an object cache that has <i>expired</i> . The time to live value has gone to zero and the object may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.
NIS_NAMEUNREACHABLE	A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the HARD_LOOKUP flag.
NIS_UNKNOWNOBJ	The object returned is of an unknown type.
NIS_TRYAGAIN	The server connected to was too busy to handle your request. For the <i>add</i> , <i>remove</i> , and <i>modify</i> operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of nis_list() if the client specifies a callback and the server does not have enough resources to handle the callback.
NIS_SYSTEMERROR	A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.
NIS_NOT_ME	A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

	NIS_NOMEMORY	Generally a fatal result. It means that the service ran out of heap space.
	NIS_NAMEEXISTS	An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.
	NIS_NOTMASTER	An attempt was made to update the database on a replica server.
	NIS_INVALIDOBJ	The object pointed to by <i>obj</i> is not a valid NIS+ object.
	NIS_BADNAME	The name passed to the function is not a legal NIS+ name.
	NIS_LINKNAMEERROR	The name passed resolved to a <i>LINK</i> type object and the contents of the link pointed to an invalid name.
	NIS_NOTSAMEOBJ	An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.
	NIS_NOSUCHNAME	This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.
	NIS_NOSUCHTABLE	The named table does not exist.
	NIS_MODFAIL	The attempted modification failed.
	NIS_FOREIGNNS	The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY , which contains the type of namespace and contact information for a server within that namespace.
	NIS_RPCERROR	This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog (3) message indicating why the RPC request failed.
ENVIRONMENT	NIS_PATH	If the flag EXPAND_NAME is set, this variable is the search path used by nis_lookup() .
SUMMARY OF TRUSTED SOLARIS CHANGES		To succeed, nis_add() , nis_modify() , and nis_remove() must inherit the PAF_TRUSTED_PATH attribute.
SEE ALSO	nis_error (3N), nis_objects (3N), nis_server (3N), nis_subr (3N), nis_tables (3N)	
NOTES	You cannot modify the name of an object if that modification would cause the object to reside in a different domain.	

You cannot modify the schema of a table object.

NAME	nis_ping, nis_checkpoint – misc NIS+ log administration functions
SYNOPSIS	<pre>cc [flag ...] file... -lnsl [library...] #include <rpcsvc/nis.h> void nis_ping(const nis_name dirname, const u_long utime, const nis_object *dirobj); nis_result *nis_checkpoint(const nis_name dirname);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter <i>dirname</i> identifies the directory with the change. If the parameter <i>dirobj</i> is NULL, this function looks up the directory object for <i>dirname</i> and uses the list of replicas it contains. The parameter <i>utime</i> contains the timestamp of the last change made to the directory. This timestamp is used by the replicas when retrieving updates made to the directory. The effect of calling nis_ping() is to schedule an update on the replica. A short time after a ping is received, typically about two minutes, the replica compares the last update time for its databases to the timestamp sent by the ping. If the ping timestamp is later, the replica establishes a connection with the master server and request all changes from the log that occurred after the last update that it had recorded in its local log. To succeed, nis_ping() must inherit the PAF_TRUSTED_PATH attribute. nis_checkpoint() is used to force the service to checkpoint information that has been entered in the log but has not been checkpointed to disk. When called, this function checkpoints the database for each table in the directory, the database containing the directory and the transaction log. Care should be used in calling this function since directories that have seen a lot of changes may take several minutes to checkpoint. During the checkpointing process, the service will be unavailable for updates for all directories that are served by this machine as master. nis_checkpoint() returns a pointer to a <i>nis_result</i> structure (described in nis_tables(3N)). This structure should be freed with nis_freeresult() (see nis_names(3N)). The only items of interest in the returned result are the status value and the statistics.
SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, nis_ping() , must inherit the PAF_TRUSTED_PATH attribute.
SEE ALSO	nislog(1M) , nis_names(3NTSOL) , nis_tables(3NTSOL) , nisfiles(4)

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flag ...] file... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(const nis_name dirname, const nis_server *machine); nis_error nis_rmdir(const nis_name dirname, const nis_server *machine); nis_error nis_servstate(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result); nis_error nis_stats(const nis_server *machine, const nis_tag *tags, const int numtags, nis_tag **result); void nis_freetags(nis_tag *tags, const int numtags); nis_server **nis_getservlist(const nis_name dirname); void nis_freeservlist(nis_server **machines);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions provide a variety of services for NIS+ applications. nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i>, on a server, <i>machine</i>. If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the nis_server structure, refer to nis_objects(3N). To succeed, nis_mkdir() must inherit the PAF_TRUSTED_PATH attribute. nis_rmdir() is used to delete the directory, <i>dirname</i>, from the specified machine. The <i>machine</i> parameter cannot be NULL. For a description of the nis_server structure, refer to nis_objects(3N). To succeed, nis_rmdir() must inherit the PAF_TRUSTED_PATH attribute. nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, nis_servstate() must inherit the PAF_TRUSTED_PATH attribute. The nis_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see nisstat(1M). nis_servstate() and nis_stats() use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <rpcsvc/nis_tags.h> which is included in <rpcsvc/nis.h>. Because these tags can and do vary between implementations of the NIS+ service, it is best to consult this file for the supported list. Passing unrecognized tags to a server will result in their <i>tag_value</i> member being set to the string “unknown.” Both

of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to **NULL**. The *tag_value* pointers points to allocated string memory which contains the results. Use **nis_freetags()** to free the tag structure.

nis_getservlist() returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the **nis_server** structure, refer to **nis_objects(3N)**. **nis_freeservlist()** frees the list of servers returned by **nis_getservlist()**. Note that this is the only legal way to free that list.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

To succeed, **nis_mkdir()**, **nis_rmdir()**, and **nis_servstate()** must inherit the PAF_TRUSTED_PATH attribute.

SEE ALSO

nisstat(1M), **nis_names(3NTSOL)**, **nis_objects(3N)**, **nis_subr(3N)**

NAME	<code>nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flag ...] file... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(const nis_name name, const u_long flags, int (*callback)(const nis_name table_name, const nis_object *object, const void *userdata), const void *userdata); nis_result *nis_add_entry(const nis_name table_name, const nis_object *object, const u_long flags); nis_result *nis_remove_entry(const nis_name name, const nis_object *object, const u_long flags); nis_result *nis_modify_entry(const nis_name name, const nis_object *object, const u_long flags); nis_result *nis_first_entry(const nis_name table_name); nis_result *nis_next_entry(const nis_name table_name, const netobj *cookie); void nis_freeresult(nis_result *result);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	These functions are used to search and modify NIS+ tables. nis_list() is used to search a table in the NIS+ namespace. nis_first_entry() and nis_next_entry() are used to enumerate a table one entry at a time. nis_add_entry(), nis_remove_entry(), and nis_modify_entry() are used to change the information stored in a table. nis_freeresult() is used to free the memory associated with the nis_result structure. Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[']' characters. Indexed names have the following form: <pre>[colname=value, ...],tablename</pre> The list function, nis_list(), takes an indexed name as the value for the <i>name</i> parameter. Here, the tablename should be a fully qualified NIS+ name unless the EXPAND_NAME flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically ORing together one or more flags from the following list. FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a LINK type object (see nis_objects(3N)), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error NIS_NOTSEARCHABLE will be returned. FOLLOW_PATH This flag specifies that if the entry is not found within this table, the list

operation should follow the path specified in the table object. When used in conjunction with the **ALL_RESULTS** flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the **FOLLOW_LINKS** flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.

HARD_LOOKUP

This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as **NIS_NOTFOUND**).

ALL_RESULTS

This flag can only be used in conjunction with **FOLLOW_PATH** and a callback function. When specified, it forces all of the tables in the path to be searched. If *name* does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.

NO_CACHE

This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.

MASTER_ONLY

This flag is even stronger than **NO_CACHE** in that it specifies that the client library should *only* get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the **HARD_LOOKUP** flag, this will block the list operation until the master server is up and available.

EXPAND_NAME

When specified, the client library will attempt to expand a partially qualified name by calling **nis_getnames()** (see **nis_local_names(3N)**) which uses the environment variable **NIS_PATH**.

RETURN_RESULT

This flag is used to specify that a copy of the returning object be returned in the **nis_result** structure if the operation was successful.

The third parameter to **nis_list()**, *callback*, is an optional pointer to a function that will process the **ENTRY** type objects that are returned from the search. If this pointer is **NULL**, then all entries that match the search criteria are returned in the *nis_result* structure, otherwise this function will be called once for each entry returned. When called, this function should return **0** when additional objects are desired and **1** when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state

information or other relevant data that the callback function might need to process the entries.

nis_add_entry() will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The **ADD_OVERWRITE** flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the **ADD_OVERWRITE** flag, this function will fail with the error **NIS_PERMISSION** if the existing object does not allow modify privileges to the client.

If the flag **RETURN_RESULT** has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, **nis_add_entry()** must inherit the PAF_TRUSTED_PATH attribute.

nis_remove_entry() removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an **NIS_NOTSAMEOBJ** error. If an object is passed with this function, the search criteria in name is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the flags variable is null, and the search criteria does not uniquely identify an entry, the **NIS_NOTUNIQUE** error is returned and the operation is aborted. If the flag parameter **REM_MULTIPLE** is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the **REM_MULTIPLE** flag will remove all entries in a table.

To succeed, **nis_remove_entry()** must inherit the PAF_TRUSTED_PATH attribute.

nis_modify_entry() modifies an object identified by *name*. The parameter *object* should point to an entry with the **EN_MODIFIED** flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object*: **zo_owner**, **zo_group**, and **zo_access**.

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag **MOD_SAMEOBJ** then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the **NIS_NOTSAMEOBJ** error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag **RETURN_RESULT** has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, **nis_modify_entry()** must inherit the PAF_TRUSTED_PATH attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name*. If a search criteria is present in *name* it is ignored. The value of *cookie* within the **nis_result** structure must be copied by the caller into local storage and passed as an argument to **nis_next_entry()**.

nis_next_entry() retrieves the “next” entry from a table specified by *table_name*. The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to **nis_next_entry()** there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error **NIS_CHAINBROKEN** is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type **nis_result**:

```
struct nis_result {
    nis_error  status;
    struct {
        u_int      objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    u_long      zticks;
    u_long      dticks;
    u_long      aticks;
    u_long      cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function **nis_sperrno()** (see **nis_error(3N)**).

The **objects** structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to **nis_freeresult()** (see **nis_names(3N)**). If you need to keep a copy of one or more objects, they can be copied with the function **nis_clone_object()** and freed with the function **nis_destroy_object()** (see **nis_server(3N)**).

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one’s data organization for faster access and to compare different database implementations (see **nis_db(3N)**).

- zticks* The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any “accelerators” or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE	The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.
NIS_BADNAME	The name passed to the function is not a legal NIS+ name.
NIS_BADREQUEST	A problem was detected in the request structure passed to the client library.
NIS_CACHEEXPIRED	The entry returned came from an object cache that has <i>expired</i> . This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.
NIS_CBERROR	An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.
NIS_CBRESULTS	Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.
NIS_FOREIGNNS	The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY . The returned object contains the type of namespace and contact information for a server within that namespace.
NIS_INVALIDOBJ	The object pointed to by <i>object</i> is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.
NIS_LINKNAMEERROR	The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.
NIS_MODFAIL	The attempted modification failed for some reason.

NIS_NAMEEXISTS	An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.
NIS_NAMEUNREACHABLE	This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.
NIS_NOCALLBACK	The server was unable to contact the callback service on your machine. This results in no data being returned.
NIS_NOMEMORY	Generally a fatal result. It means that the service ran out of heap space.
NIS_NOSUCHNAME	This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.
NIS_NOSUCHTABLE	The named table does not exist.
NIS_NOT_ME	A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.
NIS_NOTFOUND	No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the nis_remove() . If the FOLLOW_PATH flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.
NIS_NOTMASTER	A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the /var/nis/NIS_SHARED_DIRCACHE file will need to have their cache managers restarted (use nis_cachemgr -i) to flush this cache.
NIS_NOTSAMEOBJ	An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.
NIS_NOTSEARCHABLE	The table name resolved to a NIS+ object that was not searchable.
NIS_PARTIAL	This result is similar to NIS_NOTFOUND except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other

		local policy.
	NIS_RPCERROR	This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog (3) message indicating why the RPC request failed.
	NIS_S_NOTFOUND	The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.
	NIS_S_SUCCESS	Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.
	NIS_SUCCESS	The request was successful.
	NIS_SYSTEMERROR	Some form of generic system error occurred while attempting the request. Check the syslog (3) record for error messages from the server.
	NIS_TOOMANYATTRS	The search criteria passed to the server had more attributes than the table had searchable columns.
	NIS_TRYAGAIN	The server connected to was too busy to handle your request. add_entry() , remove_entry() , and modify_entry() return this error when the master server is currently updating its internal state. It can be returned to nis_list() when the function specifies a callback and the server does not have the resources to handle callbacks.
	NIS_TYPEMISMATCH	An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.
ENVIRONMENT	NIS_PATH	When set, this variable is the search path used by nis_list() if the flag EXPAND_NAME is set.
SUMMARY OF TRUSTED SOLARIS CHANGES		To succeed, nis_add_entry() , nis_remove_entry() , and nis_modify_entry() must inherit the PAF_TRUSTED_PATH attribute.
SEE ALSO		niscat (1), niserror (1), nismatch (1), nis_cachemgr (1MTSOL), nis_error (3N), nis_local_names (3N), nis_names (3NTSOL), nis_objects (3N), syslog (3)
WARNINGS		Use the flag HARD_LOOKUP carefully since it can cause the application to block indefinitely during a network partition.

NOTES

The path used when the flag **FOLLOW_PATH** is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling **nis_list()** with a callback from within a list callback function is not currently supported.

There are currently no known methods for **nis_first_entry()** and **nis_next_entry()** to get their answers from only the master server.

NAME	plock – lock or unlock into memory process, text, or data												
SYNOPSIS	<pre>#include <sys/lock.h> int plock(int op);</pre>												
DESCRIPTION	plock() allows the calling process to lock or unlock into memory its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock). Locked segments are immune to all routine swapping. The effective user ID of the calling process must be super-user to use this call. plock() performs the function specified by <i>op</i>: <table> <tr> <td>PROCLCK</td><td>Lock text and data segments into memory (process lock).</td></tr> <tr> <td>TXTLCK</td><td>Lock text segment into memory (text lock).</td></tr> <tr> <td>DATLOCK</td><td>Lock data segment into memory (data lock).</td></tr> <tr> <td>UNLOCK</td><td>Remove locks.</td></tr> </table>	PROCLCK	Lock text and data segments into memory (process lock).	TXTLCK	Lock text segment into memory (text lock).	DATLOCK	Lock data segment into memory (data lock).	UNLOCK	Remove locks.				
PROCLCK	Lock text and data segments into memory (process lock).												
TXTLCK	Lock text segment into memory (text lock).												
DATLOCK	Lock data segment into memory (data lock).												
UNLOCK	Remove locks.												
RETURN VALUES	Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of –1 is returned and errno is set to indicate the error.												
ERRORS	plock() fails and does not perform the requested operation if one or more of the following are true: <table> <tr> <td>EAGAIN</td><td>Not enough memory.</td></tr> <tr> <td>EINVAL</td><td><i>op</i> is equal to PROCLCK and a process lock, a text lock, or a data lock already exists on the calling process.</td></tr> <tr> <td>EINVAL</td><td><i>op</i> is equal to TXTLCK and a text lock, or a process lock already exists on the calling process.</td></tr> <tr> <td>EINVAL</td><td><i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.</td></tr> <tr> <td>EINVAL</td><td><i>op</i> is equal to UNLOCK and no lock exists on the calling process.</td></tr> <tr> <td>EPERM</td><td>The process does not have sufficient privilege.</td></tr> </table>	EAGAIN	Not enough memory.	EINVAL	<i>op</i> is equal to PROCLCK and a process lock, a text lock, or a data lock already exists on the calling process.	EINVAL	<i>op</i> is equal to TXTLCK and a text lock, or a process lock already exists on the calling process.	EINVAL	<i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.	EINVAL	<i>op</i> is equal to UNLOCK and no lock exists on the calling process.	EPERM	The process does not have sufficient privilege.
EAGAIN	Not enough memory.												
EINVAL	<i>op</i> is equal to PROCLCK and a process lock, a text lock, or a data lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to TXTLCK and a text lock, or a process lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to DATLOCK and a data lock, or a process lock already exists on the calling process.												
EINVAL	<i>op</i> is equal to UNLOCK and no lock exists on the calling process.												
EPERM	The process does not have sufficient privilege.												
SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, plock() requires PRIV_SYS_CONFIG in its set of effective privileges.												
SEE ALSO	exec(2TSOL) , exit(2) , fork(2TSOL) , memcntl(2) , mlock(3CTSOL) , mlockall(3CTSOL)												
NOTES	mlock(3C) and mlockall(3C) are the preferred interfaces to process locking.												

NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege ID to its name or a privilege name to its number	
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/priv.h> priv_t str_to_priv(const char *priv_name) char *priv_to_str(const priv_t priv_id) char *str_to_priv_set(const char *priv_names, priv_set_t *priv_set, const char *separators) char *priv_set_to_str(priv_set_t *priv_set, char separator, char *buffer, int *buflen) char *get_priv_text(const priv_t priv_id)</pre>	
AVAILABILITY	SUNWtsolu	
MT CLASS	MT-safe	
DESCRIPTION	priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by <i>priv_id</i>. If <i>priv_id</i> is an undefined privilege ID, the integer ordinal of <i>priv_id</i> is returned. If <i>priv_id</i> is greater than TSOL_MAX_PRIV, the maximum allowable privilege ID, a NULL is returned. str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name <i>priv_name</i>. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. priv_set_to_str() appends the name of each privilege in <i>priv_set</i> to a string to which the user-supplied <i>buffer</i> of length <i>buflen</i> points. Privilege names are separated by the <i>separator</i> character. Integer ordinals name the undefined privileges found in the privilege set. String none identifies an empty privilege set; and all, a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort. Based on the token separators (<i>separators</i>), str_to_priv_set() breaks the <i>priv_names</i> string into tokens to be translated into a privilege set. Token none is translated to an empty privilege set; token all, to a full privilege set. The presence of token none overrides whatever precedes it. For example, the string file_mac_read,file_mac_write,none,proc_nofloat produces the same result as proc_nofloat alone. The constructed privilege set is stored in the priv_set_t buffer to which <i>priv_set</i> points. get_priv_text() returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i>.	
RETURN VALUES	priv_to_str()	Returns a pointer to the translated privilege name string. The function returns NULL and sets errno on failure.
	str_to_priv()	Returns the numeric privilege ID. The function returns -1 and sets

	errno on failure.
	priv_set_to_str() Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets errno on failure. This function returns -1 if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
	str_to_priv_set() Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.
ERROR	priv_to_str() may fail for this reason: EINVAL The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code>. priv_set_to_str() may fail for this reason: EFAULT The specified <i>priv_set</i> is an invalid address. str_to_priv() may fail for one of these reasons: EINVAL The specified <i>priv_name</i> does not match any of the defined privilege names. EFAULT The specified <i>priv_name</i> is an invalid address.
NOTES	To use these routines, the program must be loaded with the Trusted Solaris library libtsol or libtsol.so .
SEE ALSO	priv_desc(4TSOL) priv_name(4TSOL)

NAME	randomword – generate random pronounceable password
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/tsol.h> int randomword(char *word, char *hyphenated_word, const unsigned short minlen, const unsigned short maxlen, const unsigned char *seed);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Unsafe
DESCRIPTION	randomword() generates random pronounceable passwords using the FIPS 181 algorithm. Upon successful completion, <i>word</i> is replaced with a new password with a length between <i>minlen</i> and <i>maxlen</i> inclusive. <i>hyphenated_word</i> is a hyphenated version of <i>word</i> showing its pronunciation. If <i>seed</i> is non-null, it is a random number seed of eight significant characters. A good choice is the user's old password. Successive calls to randomword by the same program should pass a null pointer for <i>seed</i> to produce new random passwords using the initial <i>seed</i>.
RETURN VALUE	randomword returns: –1 If <i>minlen</i> > <i>maxlen</i> or <i>seed</i> has not been set. 0 If <i>maxlen</i> is zero (0). > 0 Length of the password <i>word</i> generated.
EXAMPLES	<pre>char password[10]; char hyphen_password[20]; char seed[9]; int len; int i; printf("Please enter old password: "); fgets(seed, 9, stdin); len = randomword(password, hyphen_password, 6, 8, seed); printf("password %s is pronounced %s\n", password, hyphen_password); for (i = 1; i < 5; i++) { len = randomword(password, hyphen_password, 6, 8, (unsigned char *) 0); printf("password %s is pronounced %s\n", password, hyphen_password); }</pre>

SEE ALSO

Federal Information Processing Standards Publication 181, *Automated Password Generator*, 5 October 1993

NAME	resolver, res_init, res_mkquery, res_send, res_search, dn_comp, dn_expand – resolver routines																
SYNOPSIS	<pre>cc [flag ...] file ... -lresolv -lsocket -lnsl [library ...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, char *dname, int class, int type, char *data, int datalen, struct rrec *newrr, char *buf, int buflen); int res_send(char *msg, int msglen, char *answer, int anslen); int res_search(char *dname, int class, int type, uchar *answer, int anslen); int dn_comp(char *exp_dn, char *comp_dn, int length, char **dnptrs, char **lastdnptr); int dn_expand(char *msg, char *comp_dn, char exp_dn, int msglen, int length);</pre>																
MT-LEVEL	Unsafe																
DESCRIPTION	These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable _res. Most of the values have reasonable defaults and can be ignored. Options are a simple bit mask and are OR-ed in to enable. Options stored in _res.options are defined in <resolv.h> and are as follows. <table> <tr> <td>RES_INIT</td><td>True if the initial name server address and default domain name are initialized (that is, res_init() has been called).</td></tr> <tr> <td>RES_DEBUG</td><td>Print debugging messages.</td></tr> <tr> <td>RES_AAONLY</td><td>Accept authoritative answers only. res_send() will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.</td></tr> <tr> <td>RES_USEVC</td><td>Use TCP connections for queries instead of UDP.</td></tr> <tr> <td>RES_PRIMARY</td><td>Query primary server only.</td></tr> <tr> <td>RES_IGNTC</td><td>Unused currently (ignore truncation errors, that is, do not retry with TCP).</td></tr> <tr> <td>RES_RECURSE</td><td>Set the recursion desired bit in queries. This is the default. res_send() does not do iterative queries and expects the name server to handle recursion.</td></tr> <tr> <td>RES_DEFNAMES</td><td>Append the default domain name to single label queries. This is the default.</td></tr> </table>	RES_INIT	True if the initial name server address and default domain name are initialized (that is, res_init() has been called).	RES_DEBUG	Print debugging messages.	RES_AAONLY	Accept authoritative answers only. res_send() will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.	RES_USEVC	Use TCP connections for queries instead of UDP.	RES_PRIMARY	Query primary server only.	RES_IGNTC	Unused currently (ignore truncation errors, that is, do not retry with TCP).	RES_RECURSE	Set the recursion desired bit in queries. This is the default. res_send() does not do iterative queries and expects the name server to handle recursion.	RES_DEFNAMES	Append the default domain name to single label queries. This is the default.
RES_INIT	True if the initial name server address and default domain name are initialized (that is, res_init() has been called).																
RES_DEBUG	Print debugging messages.																
RES_AAONLY	Accept authoritative answers only. res_send() will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.																
RES_USEVC	Use TCP connections for queries instead of UDP.																
RES_PRIMARY	Query primary server only.																
RES_IGNTC	Unused currently (ignore truncation errors, that is, do not retry with TCP).																
RES_RECURSE	Set the recursion desired bit in queries. This is the default. res_send() does not do iterative queries and expects the name server to handle recursion.																
RES_DEFNAMES	Append the default domain name to single label queries. This is the default.																

RES_STAYOPEN Used with **RES_USEVC** to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_DNSRCH Search up local domain tree.

res_init() reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried.

res_mkquery() makes a standard query message and places it in *buf*. **res_mkquery()** will return the size of the query or **-1** if the query is larger than *buflen*. *op* is usually **QUERY** but can be any of the query types defined in `<arpa/nameser.h>`. *dname* is the domain name. If *dname* consists of a single label and the **RES_DEFNAMES** flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable **LOCALDOMAIN**. *newrr* is currently unused but is intended for making update messages. *class* and *type* define the class and type of query. **data* is the resource record; and *datalen* is the length of the record.

res_send() sends a query to name servers and returns an answer. It will call **res_init()** if **RES_INIT** is not set, send the query to the local name server, and handle timeouts and retries. *msg* is the query sent; *msglen* is its length. *answer* is the response returned. The length of the response is stored in *anslen*. **res_send()** returns the length of the response or **-1** if there were errors.

If the query is sent to a name server on a non-trusted host, **res_send()** and **res_search()** can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the **net_upgrade_sl**, **net_downgrade_sl**, and **net_mac_read** privileges, then **res_send()** and **res_search()** can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

res_search() formulates and sends a normal query (**QUERY**) message, and stores the response in a buffer supplied by the caller. *dname* is the domain name. *class* and *type* define the class and type of query (see `<arpa/nameser.h>`). The response is returned in the user-supplied buffer *answer*. **res_search** returns the length of *answer* in *anslen*.

res_search() will call **res_init()** if the **RES_INIT** flag is not enabled. If *dname* consists of a single label and the **RES_DEFNAMES** flag is enabled (the default), *dname* will be appended with the current domain name. If the **RES_DNSRCH** flag is enabled, **res_search()** will search up the local domain tree until an answer has been retrieved or an unrecoverable error has been encountered. **res_search()** returns the length of *answer* on success and **-1** on error. Note that **res_search()** is only useful for queries in the same name hierarchy as the local host.

dn_expand() expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or **-1** if there was an error.

dn_comp() compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or **-1** if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to the beginning of the message and the list ends with **NULL**. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by **dn_comp()** as the name is compressed. If *dnptrs* is **NULL**, do not try to compress names. If *lastdnptr* is **NULL**, do not update the list.

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, **res_send()** and **res_search()** can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the **PRIV_NET_UPGRADE_SL**, **PRIV_NET_DOWNGRADE_SL**, and **PRIV_NET_MAC_READ** privileges, then **res_send()** and **res_search()** can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

FILES

/etc/resolv.conf

SEE ALSO

in.named(1MTSOL), **nstest(1M)**, **resolv.conf(4)**

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	rpc – Library routines for remote procedure calls
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <rpc/rpc.h> #include <netconfig.h></pre>
MT-LEVEL	MT-Safe with exceptions
DESCRIPTION	These routines allow C language programs to make procedure calls on other machines across a network. First, the client sends a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. All RPC (remote procedure calls) routines require the header <rpc/rpc.h>. Routines that take a netconfig structure also require that <netconfig.h> be included. Applications using RPC and XDR routines should be linked with the libnsl library.
Multithread Considerations	In the case of multithreaded applications, the _REENTRANT flag must be defined on the command line at compilation time (-D_REENTRANT). Defining this flag enables a thread-specific version of rpc_createerrpc_clnt_create(3N). Client-side routines are MT-Safe. CLIENT handles [see rpc_clnt_create(3N)] can be shared between threads; however, in this implementation, requests by different threads are serialized; that is, the first request will receive its results before the second request is sent. Server-side routines are mostly MT-Unsafe. In this implementation, the service transport handle, SVCXPRT [see rpc_svc_create(3N)] contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. Routines that are affected by this restriction are marked as unsafe for MT applications. [See rpc_svc_calls(3N).]
Nettype	Some of the high-level RPC interface routines [such as clnt_create(), svc_create(), rpc_reg(), and rpc_call()] take a <i>nettype</i> string as one of the parameters. This string defines a class of transports that can be used for a particular application. <i>nettype</i> can be one of the following: netpath Choose from the transports that have been indicated by their token names in the NETPATH environment variable. If it is unset or NULL, NETPATH defaults to visible. netpath is the default <i>nettype</i>. visible Choose the transports that have the visible flag (v) set in the /etc/netconfig file. circuit_v This is same as visible except that it chooses only the connection-oriented transports (semantics tpi_cots or tpi_cots_ord) from the entries in the /etc/netconfig file.

datagram_v This is same as **visible** except that it chooses only the connectionless datagram transports (semantics **tpi_clts**) from the entries in the **/etc/netconfig** file.

circuit_n This is same as **netpath** except that it chooses only the connection-oriented datagram transports (semantics **tpi_cots** or **tpi_cots_ord**).

datagram_n This is same as **netpath** except that it chooses only the connectionless datagram transports (semantics **tpi_clts**).

udp This refers to Internet UDP.

tcp This refers to Internet TCP.

If it is **NULL**, *nettype* defaults to **netpath**. The transports are tried in left-to-right order in the **NETPATH** variable or in top-to-bottom order in the **/etc/netconfig** file.

Data Structures

Some of the data structures used by the RPC package are shown in subsequent subsections.

The AUTH Structure

```
union des_block {
    struct {
        u_int32 high;
        u_int32 low;
    } key;
    char c[8];
};
typedef union des_block des_block;
extern bool_t xdr_des_block();

/*
 * Authentication info. Opaque to client.
 */
struct opaque_auth {
    enum_t    oa_flavor;    /* flavor of auth */
    caddr_t   oa_base;      /* address of more auth stuff */
    u_int     oa_length;    /* not to exceed MAX_AUTH_BYTES */
};

/*
 * Auth handle, interface to client side authenticators.
 */
typedef struct {
    struct opaque_auth  ah_cred;
    struct opaque_auth  ah_verf;
    union des_block     ah_key;
    struct auth_ops {
        void (*ah_nextverf)();
        int  (*ah_marshall)(); /* nextverf & serialize */
    };
};
```

**The CLIENT
Structure**

```

    int      (*ah_validate)(); /* validate verifier */
    int      (*ah_refresh)(); /* refresh credentials */
    void     (*ah_destroy)(); /* destroy this structure */
} *ah_ops;
caddr_t ah_private;
} AUTH;

/*
 * Client rpc handle.
 * Created by individual implementations.
 * Client is responsible for initializing auth.
 */
typedef struct {
    AUTH      *cl_auth; /* authenticator */
    struct clnt_ops {
        enum clnt_stat (*cl_call)(); /* call remote procedure */
        void (*cl_abort)(); /* abort a call */
        void (*cl_geterr)(); /* get specific error code */
        bool_t (*cl_freeres)(); /* frees results */
        void (*cl_destroy)(); /* destroy this structure */
        bool_t (*cl_control)(); /* the ioctl() of rpc */
    } *cl_ops;
    caddr_t    cl_private; /* private stuff */
    char       *cl_netid; /* network identifier */
    char       *cl_tp; /* device name */
    t6attr_t    cl_tsol_outgoing_attrsp; /* handle for outgoing security attrs */
    t6attr_t    cl_tsol_incoming_attrsp; /* handle for incoming security attrs */
    t6mask_t    cl_tsol_incoming_new_attrsp; /* mask for incoming security attrs */
} CLIENT;

```

**The SVCXPRT
Structure**

```

enum xpirt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};

/*
 * Server side transport handle
 */
typedef struct {
    int      xp_fd; /* file descriptor for the
                    server handle */
    u_short  xp_port; /* obsolete */
    struct xp_ops {
        bool_t (*xp_rcv)(); /* receive incoming requests */
    }
}

```

```

enum xpvt_stat    (*xp_stat)();           /* get transport status */
bool_t           (*xp_getargs)();         /* get arguments */
bool_t           (*xp_reply)();           /* send reply */
bool_t           (*xp_freeargs)();        /* free mem allocated
                                         for args */
void             (*xp_destroy)();         /* destroy this struct */
} *xp_ops;
int              xp_addrlen;              /* length of remote addr.
                                         Obsolete */
char             *xp_tp;                  /* transport provider device
                                         name */
char             *xp_netid;               /* network identifier */
struct netbuf     xp_ltaddr;              /* local transport address */
struct netbuf     xp_rtaddr;              /* remote transport address */
char             xp_raddr[16];           /* remote address. Obsolete */
struct opaque_auth xp_verf;               /* raw response verifier */
caddr_t          xp_p1;                   /* private: for use
                                         by svc ops */
caddr_t          xp_p2;                   /* private: for use
                                         by svc ops */
caddr_t          xp_p3;                   /* private: for use
                                         by svc lib */
int              xp_type                  /* transport type */
t6attr_t         xp_tsol_outgoing_attrsp; /* handle for outgoing security attrs */
t6attr_t         xp_tsol_incoming_attrsp; /* handle for incoming security attrs */
t6mask_t         xp_tsol_incoming_new_attrsp; /* mask for incoming security attrs */
} SVCXPRT;

```

The svc_reg Structure

```

struct svc_req {
    u_long         rq_prog;                /* service program number */
    u_long         rq_vers;                /* service protocol version */
    u_long         rq_proc;                /* the desired procedure */
    struct opaque_auth rq_cred;            /* raw creds from the wire */
    caddr_t        rq_clntcred;            /* read only cooked cred */
    SVCXPRT        *rq_xprt;              /* associated transport */
};

```

The XDR Structure

```

/*
 * XDR operations.
 * XDR_ENCODE causes the type to be encoded into the stream.
 * XDR_DECODE causes the type to be extracted from the stream.
 * XDR_FREE can be used to release the space allocated by an XDR_DECODE
 * request.
 */
enum xdr_op {

```

```

    XDR_ENCODE=0,
    XDR_DECODE=1,
    XDR_FREE=2
};
/*
 * This is the number of bytes per unit of external data.
 */
#define BYTES_PER_XDR_UNIT(4)
#define RNDUP(x) (((x) + BYTES_PER_XDR_UNIT - 1) /
    BYTES_PER_XDR_UNIT) \ * BYTES_PER_XDR_UNIT)

/*
 * A xdrproc_t exists for each data type which is to be encoded or
 * decoded. The second argument to the xdrproc_t is a pointer to a
 * opaque pointer. The opaque pointer generally points to a
 * structure of the data type to be decoded. If this points to 0,
 * then the type routines should allocate dynamic storage of the
 * appropriate size and return it.
 * bool_t (*xdrproc_t)(XDR *, caddr_t *);
 */
typedef bool_t (*xdrproc_t)();

/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the particular implementation
 */
typedef struct {
    enum xdr_op  x_op;    /* operation; fast additional param */
    struct xdr_ops {
        bool_t  (*x_getlong)();    /* get a long from underlying stream */
        bool_t  (*x_putlong)();    /* put a long to underlying stream */
        bool_t  (*x_getbytes)();   /* get bytes from underlying stream */
        bool_t  (*x_putbytes)();   /* put bytes to underlying stream */
        u_int   (*x_getpostn)();   /* returns bytes off from beginning */
        bool_t  (*x_setpostn)();   /* repositions the stream */
        long *   (*x_inline)();    /* buf quick ptr to buffered data */
        void     (*x_destroy)();    /* free privates of this xdr_stream */
    } *x_ops;
    caddr_t      x_public;         /* users' data */
    caddr_t      x_private;        /* pointer to private data */
    caddr_t      x_base;          /* private used for position info */
    int          x_handy;         /* extra private word */
} XDR;

```

Index to Routines

The following table lists RPC routines and the manual reference pages on which they are described:

RPC Routine	Manual Reference Page
auth_destroy	rpc_clnt_auth(3N)
authdes_create	rpc_soc(3N)
authdes_getucred	secure_rpc(3N)
authdes_seccreate	secure_rpc(3N)
authkerb_getucred	kerberos_rpc(3N)
authkerb_seccreate	kerberos_rpc(3N)
authnone_create	rpc_clnt_auth(3N)
authsys_create	rpc_clnt_auth(3N)
authsys_create_default	rpc_clnt_auth(3N)
authunix_create	rpc_soc(3N)
authunix_create_default	rpc_soc(3N)
callrpc	rpc_soc(3N)
clnt_broadcast	rpc_soc(3N)
clnt_call	rpc_clnt_calls(3NTSOL)
clnt_control	rpc_clnt_create(3NTSOL)
clnt_create	rpc_clnt_create(3NTSOL)
clnt_destroy	rpc_clnt_create(3NTSOL)
clnt_dg_create	rpc_clnt_create(3NTSOL)
clnt_freeres	rpc_clnt_calls(3NTSOL)
clnt_geterr	rpc_clnt_calls(3NTSOL)
clnt_pcreateerror	rpc_clnt_create(3NTSOL)
clnt_perrno	rpc_clnt_calls(3NTSOL)
clnt_perror	rpc_clnt_calls(3NTSOL)
clnt_raw_create	rpc_clnt_create(3NTSOL)
clnt_screateerror	rpc_clnt_create(3NTSOL)
clnt_serrno	rpc_clnt_calls(3NTSOL)
clnt_serror	rpc_clnt_calls(3NTSOL)
clnt_tli_create	rpc_clnt_create(3NTSOL)
clnt_tp_create	rpc_clnt_create(3NTSOL)
clnt_udpcreate	rpc_soc(3N)
clnt_vc_create	rpc_clnt_create(3NTSOL)
clntraw_create	rpc_soc(3N)
clnttcp_create	rpc_soc(3N)
clntudp_bufcreate	rpc_soc(3N)
get_myaddress	rpc_soc(3N)
getnetname	secure_rpc(3N)
host2netname	secure_rpc(3N)
key_decryptsession	secure_rpc(3N)
key_encryptsession	secure_rpc(3N)
key_gendes	secure_rpc(3N)
key_setsecret	secure_rpc(3N)

netname2host	secure_rpc(3N)
netname2user	secure_rpc(3N)
pmap_getmaps	rpc_soc(3N)
pmap_getport	rpc_soc(3N)
pmap_rmtcall	rpc_soc(3N)
pmap_set	rpc_soc(3N)
pmap_unset	rpc_soc(3N)
rac_drop	rpc_rac(3N)
rac_poll	rpc_rac(3N)
rac_recv	rpc_rac(3N)
rac_send	rpc_rac(3N)
registerpc	rpc_soc(3N)
rpc_broadcast	rpc_clnt_calls(3NTSOL)
rpc_broadcast_exp	rpc_clnt_calls(3NTSOL)
rpc_call	rpc_clnt_calls(3NTSOL)
rpc_reg	rpc_svc_calls(3NTSOL)
svc_create	rpc_svc_create(3NTSOL)
svc_destroy	rpc_svc_create(3NTSOL)
svc_dg_create	rpc_svc_create(3NTSOL)
svc_dg_enablecache	rpc_svc_calls(3NTSOL)
svc_fd_create	rpc_svc_create(3NTSOL)
svc_fds	rpc_soc(3N)
svc_freeargs	rpc_svc_reg(3NTSOL)
svc_getargs	rpc_svc_reg(3NTSOL)
svc_getcaller	rpc_soc(3N)
svc_getreq	rpc_soc(3N)
svc_getreqset	rpc_svc_calls(3NTSOL)
svc_getrpccaller	rpc_svc_calls(3NTSOL)
svc_kerb_reg	kerberos_rpc(3N)
svc_raw_create	rpc_svc_create(3NTSOL)
svc_reg	rpc_svc_calls(3NTSOL)
svc_register	rpc_soc(3N)
svc_run	rpc_svc_reg(3NTSOL)
svc_sendreply	rpc_svc_reg(3NTSOL)
svc_tli_create	rpc_svc_create(3NTSOL)
svc_tp_create	rpc_svc_create(3NTSOL)
svc_unreg	rpc_svc_calls(3NTSOL)
svc_unregister	rpc_soc(3N)
svc_vc_create	rpc_svc_create(3NTSOL)
svcerr_auth	rpc_svc_err(3N)
svcerr_decode	rpc_svc_err(3N)
svcerr_noproc	rpc_svc_err(3N)
svcerr_noprogram	rpc_svc_err(3N)
svcerr_progvers	rpc_svc_err(3N)
svcerr_systemerr	rpc_svc_err(3N)

svcerr_weakauth	rpc_svc_err(3N)
svcf_create	rpc_soc(3N)
svccraw_create	rpc_soc(3N)
svctcp_create	rpc_soc(3N)
svcudp_bufcreate	rpc_soc(3N)
svcudp_create	rpc_soc(3N)
user2netname	secure_rpc(3N)
xdr_accepted_reply	rpc_xdr(3N)
xdr_authsys_parms	rpc_xdr(3N)
xdr_authunix_parms	rpc_soc(3N)
xdr_callhdr	rpc_xdr(3N)
xdr_callmsg	rpc_xdr(3N)
xdr_opaque_auth	rpc_xdr(3N)
xdr_rejected_reply	rpc_xdr(3N)
xdr_replymsg	rpc_xdr(3N)
xprt_register	rpc_svc_calls(3NTSOL)
xprt_unregister	rpc_svc_calls(3NTSOL)

SUMMARY OF TRUSTED SOLARIS CHANGES

The **CLIENT** and **SVCXPRT** structures allow clients and servers to provide **t6attr_t** pointers to opaque structures for accessing security attributes on requests and replies. When a new **CLIENT** or **SVCXPRT** structure is created, the pointers are initialized to **NULL**. If it needs to access the security attributes, the client or server must use the **t6alloc_blk** routine to allocate attribute control structures and set the **t6attr_t** pointers in the **CLIENT** or **SVCXPRT** structure. When **clnt_destroy()** or **svc_destroy()** is used to destroy a handle, the client or server should also use **t6free_blk** to free any attribute control structures previously allocated for that handle.

FILES

/etc/netconfig

SEE ALSO

getnetconfig(3N), **getnetpath(3N)**, **kerberos_rpc(3N)**, **libt6(3NTSOL)**, **t6alloc_blk(3NTSOL)**, **t6free_blk(3NTSOL)**, **rpc_clnt_auth(3N)**, **rpc_clnt_calls(3NTSOL)**, **rpc_clnt_create(3NTSOL)**, **rpc_svc_calls(3NTSOL)**, **rpc_svc_create(3NTSOL)**, **rpc_svc_err(3N)**, **rpc_svc_reg(3NTSOL)**, **rpc_xdr(3N)**, **rpcbind(3NTSOL)**, **secure_rpc(3N)**, **xdr(3N)**, **netconfig(4)**, **rpc(4)**, **environ(5)**

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client-side calls
MT-LEVEL	MT-Safe
DESCRIPTION	RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. The clnt_call(), rpc_call(), and rpc_broadcast() routines handle the client side of the procedure call. The remaining routines deal with error handling. Some of the routines take a CLIENT handle as one of the parameters. A CLIENT handle can be created by an RPC creation routine such as clnt_create(). [See rpc_clnt_create(3NTSOL).] These routines are safe for use in multithreaded applications. CLIENT handles can be shared between threads; however, in this implementation, requests by different threads are serialized; that is, the first request will receive its results before the second request is sent. Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.
Routines	See rpc(3NTSOL) for the definition of the CLIENT data structure. #include <rpc/rpc.h> enum clnt_stat clnt_call(CLIENT *clnt, const u_long procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout); This function macro calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client-creation routine such as clnt_create(). [See rpc_clnt_create(3NTSOL).] The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address at which to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through clnt_control(). See rpc_clnt_create(3NTSOL). If the remote call succeeds, the status returned is RPC_SUCCESS; otherwise, an appropriate status is returned.

bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);

This function macro frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter *out* is the address of the results, and *outproc* is the XDR routine describing the results. This routine returns **1** if the results were successfully freed, or **0** otherwise.

void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);

This function macro copies the error structure out of the **CLIENT** handle to the structure at address *errp*.

void clnt_perrno(const enum clnt_stat stat);

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. This macro is normally used after a procedure call fails for a routine, such as **rpc_call()**, for which a **CLIENT** handle is not needed.

void clnt_perror(const CLIENT *clnt, const char *s);

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. This macro is normally used after a remote procedure call fails for a routine, such as **clnt_call()**, which requires a client handle.

char *clnt_sperrno(const enum clnt_stat stat);

Take the same arguments as **clnt_perrno()**; but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string that contains the message.

clnt_sperrno is normally used instead of **clnt_perrno** when the program does not have a standard error (as a program running as a server quite likely does not), or when the programmer does not want the message to be output with **printf()** [see **printf(3S)**], or when a message format different from that supported by **clnt_perrno** is to be used.

NOTE: Unlike **clnt_sperror** and **clnt_spcreatererror** [see **rpc_clnt_create(3NTSOL)**], **clnt_sperrno** does not return pointer to static data so the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT *clnt, const char *s);

This function is like **clnt_perror** except that (like **clnt_sperrno**) it returns a string instead of printing to standard error. However, **clnt_sperror** does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
enum clnt_stat rpc_broadcast(const u_long prognum, const u_long versnum,
const u_long procnum, const xdrproc_t inproc, const caddr_t in,
const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult,
const char *nettype);
```

Like **rpc_call()** except the call message is broadcast to all the connectionless transports specified by *nettype*. If it is **NULL**, *nettype* defaults to **netpath**. Each time it receives a response, this routine calls **eachresult()**, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to **rpc_broadcast()** except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results; and *netconf* is the netconfig structure of the transport on which the remote server responded. If **eachresult** returns **0**, **rpc_broadcast** waits for more replies; otherwise, **rpc_broadcast** returns with appropriate status.

The process must have the **PRIV_NET_BROADCAST** privilege.

Warning: Broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes. **rpc_broadcast** uses **AUTH_SYS** credentials by default. [See **rpc_clnt_auth(3N)**.]

```
enum clnt_stat rpc_broadcast_exp(const u_long prognum, const u_long versnum,
const u_long procnum, const xdrproc_t xargs, caddr_t argsp,
const xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult,
const int inittime, const int waittime, const char *nettype);
```

This function is like **rpc_broadcast** except that the initial timeout, *inittime*, and the maximum timeout, *waittime*, are specified in milliseconds. *inittime* is the initial time that **rpc_broadcast_exp** waits before resending the request. After the first resend, the retransmission interval increases exponentially until it exceeds *waittime*.

The process must have the **PRIV_NET_BROADCAST** privilege.

```
enum clnt_stat rpc_call(const char *host, const u_long prognum,
const u_long versnum, const u_long procnum, const xdrproc_t inproc,
const char *in, const xdrproc_t outproc, char *out, const char *nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address at which to place the result(s). *nettype* can be any of the values listed on **rpc(3NTSOL)**. This routine returns **RPC_SUCCESS** if it succeeds, or an appropriate status if it fails. Use the **clnt_perrno()** routine to translate failure status into error messages.

WARNING: **rpc_call** uses the first available transport belonging to the class

nettype, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The **CLIENT** structure allows a client to provide **t6attr_t** pointers to opaque structures for accessing the security attributes of a reply or request. When a new **CLIENT** structure is created, the pointers are initialized to **NULL**. If it needs to access the security attributes, the client uses the **t6alloc_blk** routine to allocate attribute-control structures and set the **t6attr_t** pointers in the **CLIENT** structure. When **clnt_destroy()** is used to destroy a client handle, the client should also use **t6free_blk** to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

printf(3S), **rpc(3NTSOL)**, **rpc_clnt_auth(3N)**, **rpc_clnt_create(3NTSOL)**, **libt6(3NTSOL)**, **t6alloc_blk(3NTSOL)**, **t6free_blk(3NTSOL)**

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles																		
MT-LEVEL	MT-Safe																		
DESCRIPTION	RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service and then sends a reply. These routines are MT-Safe. In the case of multithreaded applications, the _REENTRANT flag must be defined on the command line (-D_REENTRANT) at compilation time. When the _REENTRANT flag is defined, rpc_createerr becomes a macro that enables each thread to have its own rpc_createerr. Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.																		
Routines	See rpc(3NTSOL) for the definition of the CLIENT data structure. #include <rpc/rpc.h> bool_t clnt_control(CLIENT *clnt, const u_int req, char *info); This function macro changes or retrieves various information about a client object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, these are the supported values of <i>req</i>, their argument types, and what they do: <table><tr><td>CLSET_TIMEOUT</td><td>struct timeval *</td><td>Set total timeout.</td></tr><tr><td>CLGET_TIMEOUT</td><td>struct timeval *</td><td>Get total timeout.</td></tr></table> NOTE: If you set the timeout using clnt_control, the timeout argument passed by clnt_call is ignored in all subsequent calls. NOTE: If you set the timeout value to 0, clnt_control immediately returns an error (RPC_TIMEDOUT). Set the timeout parameter to 0 for batching calls. <table><tr><td>CLGET_FD</td><td>int *</td><td>Get the associated file descriptor.</td></tr><tr><td>CLGET_SVC_ADDR</td><td>struct netbuf *</td><td>Get servers address.</td></tr><tr><td>CLSET_FD_CLOSE</td><td>void</td><td>Close the file descriptor when destroying the client handle. [See clnt_destroy().]</td></tr><tr><td>CLSET_FD_NCLOSE</td><td>void</td><td>Do not close the file descriptor when destroying the client handle.</td></tr></table>	CLSET_TIMEOUT	struct timeval *	Set total timeout.	CLGET_TIMEOUT	struct timeval *	Get total timeout.	CLGET_FD	int *	Get the associated file descriptor.	CLGET_SVC_ADDR	struct netbuf *	Get servers address.	CLSET_FD_CLOSE	void	Close the file descriptor when destroying the client handle. [See clnt_destroy() .]	CLSET_FD_NCLOSE	void	Do not close the file descriptor when destroying the client handle.
CLSET_TIMEOUT	struct timeval *	Set total timeout.																	
CLGET_TIMEOUT	struct timeval *	Get total timeout.																	
CLGET_FD	int *	Get the associated file descriptor.																	
CLGET_SVC_ADDR	struct netbuf *	Get servers address.																	
CLSET_FD_CLOSE	void	Close the file descriptor when destroying the client handle. [See clnt_destroy() .]																	
CLSET_FD_NCLOSE	void	Do not close the file descriptor when destroying the client handle.																	

CLGET_VERS	unsigned long *	Get the RPC program's version number associated with the client handle.
CLSET_VERS	unsigned long *	Set the RPC program's version number associated with the client handle assuming that the RPC server for this new version is still listening at the address of the previous version.
CLGET_XID	unsigned long *	Get the XID of the previous remote procedure call.
CLSET_XID	unsigned long *	Set the XID of the next remote procedure call.

These operations are valid for connectionless transports only:

CLSET_RETRY_TIMEOUT struct timeval * Set the retry timeout.

CLGET_RETRY_TIMEOUT struct timeval * Get the retry timeout.

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control returns **TRUE** on success or **FALSE** on failure.

CLIENT *clnt_create(const char *host, const u_long prognum, const u_long versnum, const char *nettype);

This is a generic client-creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host on which the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left-to-right order in **NETPATH** variable or in top-to-bottom order in the netconfig database.

clnt_create tries all the transports of the *nettype* class available from the **NETPATH** environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using **clnt_control**. This routine returns **NULL** if it fails. The **clnt_pcreateerror** routine can be used to print the reason for failure.

NOTE: **clnt_create** returns a valid client handle even if the particular version number supplied to **clnt_create** is not registered with the **rpcbind** service. This mismatch will be discovered by a **clnt_call** later. [See **rpc_clnt_calls(3NTSOL)**.]

CLIENT *clnt_create_timed(const char *host, const u_long prognum, const u_long versnum, const char *nettype, const struct timeval *timeout);

This generic client-creation routine is similar to **clnt_create** but has the additional parameter *timeout* that specifies the maximum amount of time allowed for each

transport class tried. In all other respects, the **clnt_create_timed** call behaves exactly like the **clnt_create** call.

```
CLIENT *clnt_create_vers(const char *host, const u_long prognum,  
u_long *vers_outp, const u_long vers_low, const u_long vers_high,  
char *nettype);
```

This generic client-creation routine is similar to **clnt_create** but also checks for the version availability. *host* identifies the name of the remote host on which the server is located. *nettype* indicates the class transport protocols to be used. If successful, the routine returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server; *vers_outp* is set to this value. That is, after a successful return, *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server, then the routine fails and returns **NULL**. A default timeout is set and can be modified using **clnt_control**. This routine returns **NULL** if it fails. The **clnt_pcreateerror** routine can be used to print the reason for failure.

NOTE: **clnt_create** returns a valid client handle even if the particular version number supplied to **clnt_create** is not registered with the **rpcbind** service. This mismatch will be discovered by a **clnt_call** later. [See **rpc_clnt_calls**(3NTSOL).] However, **clnt_create_vers** does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
void clnt_destroy(CLIENT *clnt);
```

This function macro destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling **clnt_destroy**. If the RPC library opened the associated file descriptor, or if **CLSET_FD_CLOSE** was set using **clnt_control**, the file descriptor will be closed.

The caller should call **auth_destroy(clnt->cl_auth)** (before calling **clnt_destroy**) to destroy the associated AUTH structure. [See **rpc_clnt_auth**(3N).]

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr,  
const u_long prognum, const u_long versnum, const u_int sendsz,  
const u_int recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message at intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call**. [See **clnt_call** in **rpc_clnt_calls**(3NTSOL).] The retry time out and the total time out periods can be changed using **clnt_control**. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of **0** choose suitable defaults. This routine returns **NULL**

if it fails.

void clnt_pcreateerror(const char *s);

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

CLIENT *clnt_raw_create(const u_long prognum, const u_long versnum);

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space. [See **svc_raw_create** in **rpc_svc_create(3NTSOL)**.] This allows simulation of RPC and measurement of RPC overheads, such as round-trip times, without any kernel or networking interference. This routine returns **NULL** if it fails. **clnt_raw_create** should be called after **svc_raw_create**.

char *clnt_screateerror(const char *s);

This routine is like **clnt_pcreateerror** except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

WARNING: Returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

CLIENT *clnt_tli_create(const int fildes, const struct netconfig *netconf, const struct netbuf *svcaddr, const u_long prognum, const u_long versnum, const u_int sendsz, const u_int recvsz);

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is **NULL** and is connection-oriented, the file descriptor is assumed to be connected. For connectionless transports, if *svcaddr* is **NULL**, **RPC_UNKNOWNADDR** error is set. *fildes* is a file descriptor that may be open, bound, and connected. If it is **RPC_ANYFD**, it opens a file descriptor on the transport specified by *netconf*. If *fildes* is **RPC_ANYFD** and *netconf* is **NULL**, a **RPC_UNKNOWNPROTO** error is set. If *fildes* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of **0** choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), **clnt_tli_create** calls appropriate client-creation routines. This routine returns **NULL** if it fails. The **clnt_pcreateerror** routine can be used to print the reason for failure. The remote **rpcbind** service [see **rpcbind(1MTSOL)**] is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char *host, const u_long prognum,  
const u_long versnum, const struct netconfig *netconf);
```

This routine is like **clnt_create** except **clnt_tp_create** tries only one transport specified through *netconf*.

clnt_tp_create creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using **clnt_control** calls. The remote **rpcbind** service on the host *host* is consulted for the address of the remote service. This routine returns **NULL** if it fails. The **clnt_pcreateerror** routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char *host, const u_long prognum,  
const u_long versnum, const struct netconfig *netconf, const struct timeval *  
timeout );
```

This routine is like **clnt_tp_create** except **clnt_tp_create_timed** has the extra parameter *timeout*, which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the **clnt_tp_create_timed** call behaves exactly like the **clnt_tp_create** call.

```
CLIENT *clnt_vc_create(const int fildes, const struct netbuf *svcaddr,  
const u_long prognum, const u_long versnum, const u_int sendsz,  
const u_int recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. The user may use the *sendsz* and *recvsz* parameters to specify the size of the send and receive buffers; values of **0** choose suitable defaults. This routine returns **NULL** if it fails.

The address *svcaddr* should not be **NULL** and should point to the actual address of the remote program. **clnt_vc_create** does not consult the remote **rpcbind** service for this information.

```
struct rpc_createerr rpc_createerr;
```

This global variable is set by any RPC client handle-creation routine that fails. **clnt_pcreateerror** uses this variable to print the reason for the failure.

In multithreaded applications, **rpc_createerr** becomes a macro that enables each thread to have its own **rpc_createerr**.

SUMMARY OF TRUSTED SOLARIS CHANGES

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The **CLIENT** structure allows a client to provide **t6attr_t** pointers to opaque structures for accessing the security attributes of a reply or request. When a new **CLIENT** structure is created, the pointers are initialized to **NULL**. If it needs to access the security attributes, the client must use the **t6alloc_blk** routine to allocate attribute-control structures and set

the **t6attr_t** pointers in the **CLIENT** structure. When **clnt_destroy** is used to destroy a client handle, the client should also use **t6free_blk** to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

rpc(3NTSOL), **rpc_clnt_auth**(3N), **rpc_clnt_calls**(3NTSOL), **rpcbind**(1MTSOL)
libt6(3NTSOL), **t6alloc_blk**(3NTSOL), **t6free_blk**(3NTSOL)

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_pollset, svc_run, svc_sendreply – Library routines for RPC servers
MT-LEVEL	See NOTES .
DESCRIPTION	These routines are part of the RPC library that allows C language programs to make procedure calls on other machines across the network. These routines are associated with the server side of the RPC mechanism. Some of them are called by the server-side dispatch function; others [such as svc_run()] are called when the server is initiated. In the current implementation, the service transport handle SVCXPRT contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the Automatic or User MT modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. In these circumstances, some routines that would otherwise be unsafe become safe. These are marked as such. Also marked are routines that are unsafe for MT application and are not to be used by such applications. Programs can retrieve the security attributes of incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.
Routines	See rpc(3NTSOL) for the definition of the SVCXPRT data structure. #include <rpc/rpc.h> int svc_dg_enablecache(SVCXPRT *xpvt, const unsigned long cache_size); This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. There is no way to disable caching once enabled. This routine returns 1 if space necessary for a cache of the given size was successfully allocated or 0 otherwise. This function is safe in MT applications. int svc_done(SVCXPRT *xpvt); This function frees resources allocated to service a client request directed to the service endpoint <i>xpvt</i>. This call pertains only to servers executing in the User MT mode. In the User MT mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After svc_done() is invoked, the service endpoint <i>xpvt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the rpc_control() call.

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

void svc_exit(void);

This function, when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes **svc_run()** to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the **rpc_svc_create(3NTSOL)** functions or through **xprt_register(3N)**.

svc_exit() has global scope and ends all RPC server activity.

fd_set svc_fdset;

This global variable reflects the RPC server's read file-descriptor bit mask. This is of interest only if service implementors do not call **svc_run()** but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to **svc_getreqset()** or any creation routines. Do not pass its address to **select(3C)**! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the user MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

bool_t svc_freeargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);

This function macro frees any data allocated by the **RPC/XDR** system when it decoded the arguments to a service procedure using **svc_getargs()**. This routine returns **TRUE** if the results were successfully freed or **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

bool_t svc_getargs(const SVCXPRT *xprt, const xdrproc_t inproc, caddr_t in);

This function macro decodes the arguments of an RPC request associated with the RPC service transport handle *xprt*. The parameter *in* is the address at which the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns **TRUE** if decoding succeeds or **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

void svc_getreq_common(const int fd);

This routine is called to handle a request on the given file descriptor.

void svc_getreq_poll(struct pollfd *pfdp, const int pollretval);

This routine is of interest only if a service implementor does not call **svc_run()** but instead implements custom asynchronous event processing. This routine is called when **poll(2)** has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from **poll(2)** and *pfdp* is the array of *pollfd* structures on which the **poll(2)** was done. This array is assumed to be large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

void svc_getreqset(fd_set *rdfs);

This routine is of interest only if a service implementor does not call **svc_run()** but instead implements custom asynchronous event processing. This routine is called when **select(3C)** has determined that an RPC request has arrived on some RPC file descriptors; *rdfs* is the resultant read file-descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfs* have been serviced.

This function macro is unsafe in MT applications.

struct netbuf *svc_getrpcaller(const SVCXPRT *xpvt);

This is the approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpvt*.

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, the routine waits for RPC requests to arrive and calls the appropriate service procedure using **svc_getreq_poll()** when a request arrives. This procedure is usually waiting for the **poll(2)** library call to return.

Applications executing in the Automatic or User MT modes should invoke this function exactly once. In the Automatic MT mode, this function will create threads to service client requests. In the User MT mode, this function will provide a framework for service developers to create and manage their own threads for servicing client requests.

```
bool_t svc_sendreply(const SVCXPRT *xpvt, const xdrproc_t outproc,  
const caddr_t out);
```

This routine is called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xpvt* is the request's associated transport handle; *outproc* is the XDR routine that is used to encode the results; and *out* is the address of the results. This routine returns **TRUE** if it succeeds or **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

SUMMARY OF TRUSTED SOLARIS CHANGES

The **PRIV_NET_MAC_READ** privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a **bind(3N)** call is made, a multilevel port will be created.

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The **PRIV_NET_MAC_READ** privilege affects the operation of several **rpcbind** services. If the privilege is on when a library routine calls **rpcbind** to create a mapping, a multilevel mapping is created.

The **PRIV_NET_PRIVADDR** privilege is required when a library routine calls **rpcbind** to create a mapping for a transport that uses a privileged address.

The **SVCXPRT** structure allows a server to provide **t6attr_t** pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new **SVCXPRT** structure is created, the pointers are initialized to **NULL**. If it needs to access the security attributes, the server must use the **t6alloc_blk** routine to allocate attribute-control structures and set the **t6attr_t** pointers in the **SVCXPRT** structure. When **svc_destroy()** is used to destroy a service handle, the server should also use **t6free_blk** to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

bind(3N), **rpcgen(1)**, **poll(2)**, **rpc(3NTSOL)**, **rpc_control(3N)**, **rpc_svc_create(3NTSOL)**, **rpc_svc_err(3N)**, **rpc_svc_reg(3NTSOL)**, **select(3C)**, **xprt_register(3N)**, **libt6(3NTSOL)**, **t6alloc_blk(3NTSOL)**, **t6free_blk(3NTSOL)**

NOTES

svc_dg_enablecache and **svc_getrpccaller** are safe in multithreaded applications. **svc_freeargs**, **svc_getargs**, and **svc_sendreply** are safe in MT applications utilizing the Automatic or User MT modes. **svc_getreq_common**, **svc_getreqset**, and **svc_getreq_poll** are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles
MT-LEVEL	MT-Safe
DESCRIPTION	These routines are part of the RPC library that allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling svc_run. Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.
Routines	See rpc(3NTSOL) for the definition of the SVCXPRT data structure. #include <rpc/rpc.h> bool_t svc_control(SVCXPRT *svc, const u_int req, void *info); This function changes or retrieves various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. These are the supported values of <i>req</i>, their argument types, and what they do: SVCGET_VERSQUIET If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the SVCGET_VERSQUIET request, <i>*info</i> contains an integer that describes the server's current behavior: 0 indicates normal server behavior (that is, an RPC_PROGVERSMISMATCH error will be returned); 1 indicates that the out-of-range request will be silently ignored. SVCSET_VERSQUIET If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer that is either 0 (indicating normal server behavior—an RPC_PROGVERSMISMATCH error will be returned) or 1 (indicating that the out-of-range request should be silently ignored).

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *),
               const u_long prognum, const u_long versnum, const char *nettype);
```

svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports that can be used for a particular application. The transports are tried in left-to-right order in **NETPATH** variable or in top-to-bottom order in the netconfig database. If it is **NULL**, *nettype* defaults to **netpath**.

svc_create registers itself with the **rpcbind** service. [See **rpcbind**(1MTSOL).] *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling **svc_run**. [See **svc_run()** in **rpc_svc_reg**(3NTSOL).] If it succeeds, **svc_create** returns the number of server handles it created; otherwise it returns **0** and an error message is logged.

```
void svc_destroy(SVCXPRT *xpirt);
```

This function macro destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

```
SVCXPRT *svc_dg_create(const int fildes, const u_int sendsz, const u_int recvsz);
```

This routine creates a connectionless RPC service handle and returns a pointer to it. This routine returns **NULL** if it fails and logs an error message. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are **0**, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with **rpcbind**(1MTSOL).

WARNING: Because connectionless-based RPC messages can hold only limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

```
SVCXPRT *svc_fd_create(const int fildes, const u_int sendsz, const u_int recvsz);
```

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are **0**, reasonable defaults are chosen. This routine returns **NULL** if it fails and logs an error message.

```
SVCXPRT *svc_raw_create(void);
```

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space. [See **clnt_raw_create()** in **rpc_clnt_create**(3NTSOL).] This routine allows simulation of RPC and acquisition of RPC overheads (such as round-trip times) without any kernel and

networking interference. This routine returns **NULL** if it fails and logs an error message.

NOTE: **svc_run** should not be called when the raw interface is being used.

**SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig **netconf*,
const struct t_bind **bindaddr*, const u_int *sendsz*, const u_int *rcvsvsz*);**

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is **RPC_ANYFD**, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is not null, *fildev* is bound to the address specified by *bindaddr*; otherwise *fildev* is bound to a default address chosen by the transport. In the case in which the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz*; values of **0** choose suitable defaults. This routine returns **NULL** if it fails and logs an error message. The server is not registered with the **rpcbind(1MTSOL)** service.

SVCXPRT *svc_tp_create(const void (dispatch*))(const struct svc_req *,
const SVCXPRT *), const u_long *prognum*, const u_long *versnum*,
const struct netconfig **netconf*);**

svc_tp_create creates a server handle for the network specified by *netconf* and registers itself with the **rpcbind** service. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling **svc_run**. **svc_tp_create** returns the service handle if it succeeds; otherwise **svc_tp_create** returns a **NULL** and logs an error message.

SVCXPRT *svc_vc_create(const int *fildev*, const u_int *sendsz*, const u_int *rcvsvsz*);

This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns **NULL** if it fails and logs an error message. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz*; values of **0** choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the **rpcbind(1MTSOL)** service.

SUMMARY OF TRUSTED SOLARIS CHANGES

The **PRIV_NET_MAC_READ** privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as **svc_create** binds to a transport, a multilevel transport is created.

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The **PRIV_NET_MAC_READ** privilege affects the operation of several **rpcbind** services. If the privilege is on when a library routine calls **rpcbind** to create a mapping, a multilevel mapping is created.

The **PRIV_NET_PRIVADDR** privilege is required when a library routine calls **rpcbind** to create a mapping for a transport that uses a privileged address.

The **SVCXPRT** structure allows a server to provide **t6attr_t** pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new **SVCXPRT** structure is created, the pointers are initialized to **NULL**. If it needs to access the security attributes, the server must use the **t6alloc_blk** routine to allocate attribute-control structures and set the **t6attr_t** pointers in the **SVCXPRT** structure. When **svc_destroy()** is used to destroy a service handle, the server should also use **t6free_blk** to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

bind(3NTSOL), **rpcbind(1MTSOL)**, **rpc(3NTSOL)**, **rpc_svc_calls(3NTSOL)**, **rpc_svc_err(3N)**, **rpc_svc_reg(3NTSOL)**, **libt6(3NTSOL)**, **t6alloc_blk(3NTSOL)**, **t6free_blk(3NTSOL)**

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
MT-LEVEL	MT-Safe
DESCRIPTION	These routines are a part of the RPC library that allows the RPC servers to register themselves with rpcbind() [see rpcbind(1MTSOL)], and associate the given program and version number with the dispatch function. When the RPC server receives a RPC request, the library invokes the dispatch routine with the appropriate arguments.
Routines	See rpc(3NTSOL) for the definition of the SVCXPRT data structure. #include <rpc/rpc.h> bool_t rpc_reg(u_long prognum, u_long versnum, u_long procnum, char * const(*procname) (char *arg), xdrproc_t inproc, xdrproc_t outproc, const char *nettype); Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its static result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters; <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See rpc(3NTSOL). This routine returns 0 if the registration succeeded or -1 otherwise. If the server has the PRIV_NET_MAC_READ privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the PRIV_NET_PRIVADDR privilege. int svc_reg(const SVCXPRT *xpirt, const u_long prognum, const u_long versnum, const void (*dispatch), const struct netconfig *netconf); Associate <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is NULL, the service is not registered with the rpcbind service. For example, if a service has already been registered using some other means, such as inetd [see inetd(1MTSOL)], the service will not need to be registered again. If <i>netconf</i> is not zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i>→<i>nc_netid</i>] to <i>xpirt</i>→<i>xp_ltaddr</i> is established with the local rpcbind service. The svc_reg routine returns 1 if it succeeds or 0 otherwise. If the server has the PRIV_NET_MAC_READ privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the PRIV_NET_PRIVADDR privilege. void svc_unreg(const u_long prognum, const u_long versnum); Remove from the rpcbind service all mappings of the triple [<i>prognum</i>, <i>versnum</i>, <i>all-transports</i>] to network address and all mappings within the RPC service

package of the double [*prognum*, *versnum*] to dispatch routines.

The **PRIV_NET_MAC_READ** privilege is required to delete a multilevel mapping. If the mapping being deleted is for a transport that uses a privileged address, the server must have the **PRIV_NET_PRIVADDR** privilege.

int svc_auth_reg(const int cred_flavor, const enum auth_stat (*handler));

Register the service authentication routine *handler* with the dispatch mechanism so that the routine can be invoked to authenticate RPC requests received with authentication type *cred_flavor*. This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call **svc_auth_reg** after registering the service and prior to calling **svc_run**. When needed to process an RPC credential of type *cred_flavor*, the *handler* procedure will be called with two parameters (**struct svc_req** **rqst* and **struct rpc_msg** **msg*) and is expected to return a valid **enum auth_stat** value. There is no provision to change or delete an authentication handler once registered.

The **svc_auth_reg** routine returns **0** if the registration is successful, **1** if *cred_flavor* already has an authentication handler registered for it, or **-1** otherwise.

void xpvt_register(const SVCXPRT *xpvt);

After RPC service transport handle *xpvt* is created, it is registered with the RPC service package. This routine modifies the global variable **svc_fdset**. [See **rpc_svc_calls**(3NTSOL).] Service implementors usually do not need this routine.

void xpvt_unregister(const SVCXPRT *xpvt);

Before an RPC service transport handle *xpvt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable **svc_fdset**. [See **rpc_svc_calls**(3NTSOL).] Service implementors usually do not need this routine.

SUMMARY OF TRUSTED SOLARIS CHANGES

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The **PRIV_NET_MAC_READ** privilege affects the operation of several **rpcbind** services. If the privilege is on when **rpc_reg** or **rpc_svc** is called, a multilevel mapping is created. To delete a multilevel mapping, **rpc_unreg** must be called with the privilege on.

The **PRIV_NET_PRIVADDR** privilege is required for **rpc_reg**, **rpc_svc**, or **rpc_unreg** calls that create or delete mappings for a transport that uses a privileged address.

The **PRIV_NET_SETID** privilege is required by **rpc_unreg** in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

inetd(1MTSOL), **rpcbind**(1MTSOL), **rpc**(3NTSOL), **rpc_svc_calls**(3NTSOL), **rpc_svc_create**(3NTSOL), **rpc_svc_err**(3N), **rpcbind**(3NTSOL), **select**(3C)

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
MT-LEVEL	MT-Safe
DESCRIPTION	These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind [see rpcbind (1MTSOL)] maintains a list of mappings between programs and their universal addresses.
Routines	#include <rpc/rpc.h> struct rpcblist *rpcb_getmaps(const struct netconfig *netconf, const char *host); This interface to the rpcbind service, returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings. This routine will return NULL if the remote rpcbind could not be contacted. struct tsol_rpcblist *rpcb_getallmaps(const struct netconfig *netconf, const char *host); This interface to the rpcbind service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the PRIV_NET_MAC_READ privilege, all mappings are returned regardless of their sensitivity labels. This routine will return NULL if the remote rpcbind could not be contacted. bool_t rpcb_getaddr(const u_long prognum, const u_long versnum, const struct netconfig *netconf, struct netbuf *svcaddr, const char *host); This interface to the rpcbind service finds the address of the service on <i>host</i> that is registered with program number <i>prognum</i>, version <i>versnum</i>, and speaks the transport protocol associated with <i>netconf</i>. The address found is returned in <i>svcaddr</i>. <i>svcaddr</i> should be preallocated. This routine returns TRUE if it succeeds. A return value of FALSE means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilabel mapping exists, or that the RPC system failed to contact the remote rpcbind service. In the last case, the global variable rpc_createerr [see rpc_clnt_create(3NTSOL)] contains the RPC status. If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned. bool_t rpcb_gettime(const char *host, time_t *timep); This routine returns the time on <i>host</i> in <i>timep</i>. If <i>host</i> is NULL, rpcb_gettime returns the time on its own machine. This routine returns TRUE if it succeeds or

FALSE if it fails. **rpcb_gettime()** can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

```
enum clnt_stat rpcb_rmtcall(const struct netconfig *netconf, const char *host,
    const u_long prognum, const u_long versnum, const u_long procnum,
    const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc,
    caddr_t out, const struct timeval tout, struct netbuf *svcaddr);
```

This interface to the **rpcbind** service instructs **rpcbind** on *host* to make an RPC call on your behalf to a procedure on that host. The **netconfig** structure should correspond to a connectionless transport. The parameter *svcaddr* will be modified to the server's address if the procedure succeeds. [See **rpc_call()** and **clnt_call()** in **rpc_clnt_calls(3NTSOL)** for the definitions of other parameters.]

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

NOTE: Even if the server is not running, **rpcbind** does not return any error messages to the caller. In such a case, the caller times out.

NOTE: If there is no mapping at the sensitivity label of the client and no multilevel mapping, **rpcbind** does not return any error messages to the caller. In such a case, the caller times out.

NOTE: **rpcb_rmtcall** is available only for connectionless transports.

```
bool_t rpcb_set(const u_long prognum, const u_long versnum,
    const struct netconfig *netconf, const struct netbuf *svcaddr);
```

This interface to the **rpcbind** service establishes a mapping between the triple [*prognum*, *versnum*, *netconf*→*nc_netid*] and *svcaddr* on the machine's **rpcbind** service. The value of *nc_netid* must correspond to a network identifier that is defined by the netconfig database. If the client has the **PRIV_NET_MAC_READ** privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the **PRIV_NET_PRIVADDR** privilege. This routine returns **TRUE** if it succeeds or **FALSE** otherwise. [See also **svc_reg()** in **rpc_svc_calls(3NTSOL)**.] If such an entry already exists with **rpcbind**, **rpcb_set** will fail.

```
bool_t rpcb_unset(const u_long prognum, const u_long versnum,
    const struct netconfig *netconf);
```

This interface to the **rpcbind** service destroys the mapping between the triple [*prognum*, *versnum*, *netconf*→*nc_netid*] and the address on the machine's **rpcbind** service. If *netconf* is **NULL**, **rpcb_unset()** destroys all mapping between the triple [*prognum*, *versnum*, *all-transports*] and the addresses on the machine's **rpcbind** service. The **PRIV_NET_MAC_READ** privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the **PRIV_NET_PRIVADDR** privilege. This routine returns **TRUE** if it

succeeds or **FALSE** otherwise. Only the owner of the service or a process with the **PRIV_NET_SETID** privilege can destroy the mapping. [See also **svc_unreg()** in **rpc_svc_calls(3NTSOL)**.]

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Most **rpcbind** services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The **rpcb_getallmaps** function is similar to the **rpcb_getmaps** function but returns mappings at all sensitivity labels dominated by the sensitivity label of the client.

The **PRIV_NET_MAC_READ** privilege affects the operation of several **rpcbind** services. If the privilege is on when **rpcb_getallmaps** is called, all mappings are returned regardless of their sensitivity labels. If the privilege is asserted when **rpcb_set** is called, a multilevel mapping is created. To delete a multilevel mapping, **rpcb_unset** must be called with the privilege on.

The **PRIV_NET_PRIVADDR** privilege is required for **rpcb_set** or **rpcb_unset** calls that create or delete mappings for a privileged port.

The **PRIV_NET_SETID** privilege is required by **rpcb_unset** for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

rpcbind(1MTSOL), **rpcinfo(1MTSOL)**, **rpc_clnt_calls(3NTSOL)**, **rpc_svc_calls(3NTSOL)**

NAME	sbltos, sbcltos, sbsltos, sbiltos, sbcleartos – translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> char *sbcltos(const bclabel_t *label, const int len) char *sbsltos(const bslabel_t *label, const int len) char *sbiltos(const bilabel_t *label, const int len) char *sbcleartos(const bclear_t *clearance, const int len)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Unsafe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on labels that dominate the current process' <i>Sensitivity Label</i>. These functions translate <i>Binary Labels</i> into canonical strings that are clipped to the number of printable characters specified in <i>len</i>. Clipping is required if the number of characters of the translated string is greater than <i>len</i>. Clipping is done by truncating the label on the right to two characters less than the specified number of characters. A clipped indicator, “<–”, is appended to <i>Sensitivity Labels</i> and <i>Clearances</i>, and a clipped indicator, “–>”, is prepended to <i>Information Labels</i>. The character-coded label begins with a <i>Classification</i> name separated with a single space character from the list of <i>Words</i> making up the remainder of the label. The <i>Binary Labels</i> must be of the proper defined type and dominated by the process <i>Sensitivity Label</i>. In the case of a <i>Binary CMW Label</i>, the <i>Sensitivity Label</i> portion must also dominate the <i>Information Label</i> portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping. sbcltos() translates a <i>Binary CMW Label</i> into a clipped string of the form: <i>INFORMATION LABEL [SENSITIVITY LABEL]</i> using the long form of the <i>Words</i> in the <i>Information</i> and <i>Sensitivity Labels</i>, the long form of the <i>Classification</i> name of the <i>Information Label</i>, and the short form of the <i>Classification</i> name of the <i>Sensitivity Label</i>. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. sbsltos() translates a <i>Binary Sensitivity Label</i> into a clipped string using the long form of the <i>Words</i> and the short form of the <i>Classification</i> name. If <i>len</i> is less than the minimum number of characters, three, the translation fails. sbiltos() translates a <i>Binary Information Label</i> into a clipped string using the long form of the <i>Words</i> and the long form of the <i>Classification</i> name. If <i>len</i> is less than the minimum number of characters, three, the translation fails.

sbcleartos() translates a *Binary Clearance* into a clipped string using the long form of the *Words* and the short form of the *Classification* name. If *len* is less than the minimum number of characters, three, the translation fails. The translation of a *Clearance* may not be the same as the translation of a *Sensitivity Label*. These functions use different tables of the **label_encodings** file which may contain different *Words* and constraints.

RETURN VALUES

These routines return a pointer to a statically allocated string that contains the result of the translation, or **(char *)0** if the translation fails for any reason.

EXAMPLES**sbcltos**

Assume that a *CMW Label* is:

UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]

when clipped to twelve characters it is:

->U [UN TO<-

sbsltos

Assume that a *Sensitivity Label* is:

UN TOP/MIDDLE/LOWER DRAWER

when clipped to ten characters it is:

UN TOP/M<-

sbiltos

Assume that an *Information Label* is:

UNAVAILABLE TOP/LOWER DRAWER

when clipped to sixteen characters it is:

->UNAVAILABLE TO

PROCESS ATTRIBUTES

Translation of *Admin Low* and *Admin High* labels is controlled by the label view process attribute flags. If no label view process attribute is defined, their translation is controlled by the label view configured in the **label_encodings** file.

A value of *External* specifies that *Admin Low* and *Admin High* labels are mapped to the lowest and highest labels defined in the **label_encodings** file.

A value of *Internal* specifies that the *Admin Low* and *Admin High* labels are translated to the **admin low name** and **admin high name** strings specified in the **label_encodings** file. If no such names are specified, the strings **ADMIN_LOW** and **ADMIN_HIGH** are used.

FILES

/etc/security/tsol/label_encodings

The label encodings file containing the *Classification* names, *Words*, constraints, and values for the defined labels of this system.

SEE ALSO

bcltobanner(3TSOL), **bilconjoin(3TSOL)**, **blcompare(3TSOL)**, **blinset(3TSOL)**, **blmanifest(3TSOL)**, **blminmax(3TSOL)**, **blportion(3TSOL)**, **bltcolor(3TSOL)**, **bltos(3TSOL)**, **bltype(3TSOL)**, **blvalid(3TSOL)**, **btohex(3TSOL)**, **hextob(3TSOL)**, **labelinfo(3TSOL)**, **labelvers(3TSOL)**, **stobl(3TSOL)**, **label_encodings(4TSOL)**

Trusted Solaris Developer's Guide

Trusted Solaris user's document set

Trusted Solaris administrator's document set

WARNINGS

All these functions share the same statically allocated string storage. They are not MT-Safe. Subsequent calls to any of these functions will overwrite that string with the newly translated string.

NAME	send, sendto, sendmsg – Send a message from a socket				
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> int send(int s, const char *msg, int len, int flags); int sendto(int s, const char *msg, int len, int flags, const struct sockaddr *to, int tolen); int sendmsg(int s, const struct msghdr *msg, int flags);</pre>				
MT-LEVEL	Safe				
DESCRIPTION	send(), sendto(), and sendmsg() transmit a message to another transport endpoint. send may be used only when the socket is in a <i>connected</i> state; sendto and sendmsg may be used at any time. <i>s</i> is a socket created with socket(3N). The address of the target is given by <i>to</i> with its size specified by <i>tolen</i>. The length of the message is given by <i>len</i>. If the message is too long to pass atomically through the underlying protocol, then the message is not transmitted and the call returns the error EMSGSIZE. A return value of -1 indicates locally detected errors only and does not implicitly mean that the message was not delivered. If the socket does not have enough buffer space available to hold the message being sent, send block unless the socket has been placed in nonblocking I/O mode. [See fcntl(2).] The select(3C) or poll(2) call may be used to determine when it is possible to send more data. The <i>flags</i> parameter is formed from the bitwise OR of zero or more of these options: <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion provided the underlying protocol also supports out-of-band data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.</td></tr> <tr> <td>MSG_DONTROUTE</td><td>The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.</td></tr> </table> See recv(3N) for a description of the msghdr structure.	MSG_OOB	Send “out-of-band” data on sockets that support this notion provided the underlying protocol also supports out-of-band data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion provided the underlying protocol also supports out-of-band data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.				
RETURN VALUES	Upon success, these calls return the number of bytes sent. If an error occurred, the calls return -1 and set errno to indicate the error.				

ERRORS	The calls fail if any of these conditions is true: EBADF <i>s</i> is an invalid file descriptor. EINTR The operation was interrupted by delivery of a signal before any data could be buffered to be sent. EINVAL <i>to</i><i>len</i> is not the size of a valid address for the specified address family. EMSGSIZE The socket requires that message be sent atomically, and the message was too long. ENOMEM There was insufficient memory available to complete the operation. ENOSR There were insufficient STREAMS resources available for the operation to complete. ENOTSOCK <i>s</i> is not a socket. EWouldBlock The socket is marked nonblocking and the requested operation would block.
SUMMARY OF TRUSTED SOLARIS CHANGES	If the process calling these routines possesses the PRIV_NET_REPLY_EQUAL privilege, the packets the process sends will carry the same CMW label as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.
SEE ALSO	fcntl(2TSOL), poll(2), write(2TSOL), connect(3N), getsockopt(3NTSOL), recv(3N), select(3C), socket(3N)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process
SYNOPSIS	<pre>#include <tsol/priv.h> int set_effective_priv(priv_op_t op , int privno[, priv_t priv_id, ...]) int set_permitted_priv(priv_op_t op , int privno[, priv_t priv_id, ...]) int set_inheritable_priv(priv_op_t op , int privno[, priv_t priv_id, ...])</pre>
AVAILABILITY	SUNWtsolu
MT CLASS	MT-safe
DESCRIPTION	These routines, located in the Trusted Solaris library, assign the effective, inheritable, and permitted privilege sets, respectively, for the current process. These routines provide a user-friendly interface to the system call setppriv(2TSOL). <i>op</i> is one of these operations: PRIV_ON Add the specified privilege IDs to the privilege set of the target process. PRIV_OFF Clear the specified privilege IDs from the privilege set of the target process. PRIV_SET Add the specified privilege IDs to the privilege set of the target process and clear all other privileges. <i>privno</i> indicates the count of privilege IDs that follow. The behavior of these routines is undefined if <i>privno</i> is less than zero. <i>priv_id</i> is a numerical privilege ID defined in priv_names.h. Note that if <i>op</i> is PRIV_SET and <i>privno</i> is 0, the target privilege set is initialized to the empty set.
RETURN VALUES	Upon success, these routines return 0 . Upon failure, these routines return -1 and set errno to indicate the error.
ERRORS	EINVAL The specified privilege is invalid. EPERM A specified privilege is not permitted in the asserted set of privileges.
SEE ALSO	setppriv (2TSOL)

NAME	stobl, stobcl, stobsl, stobil, stobclear – translate character-coded labels to binary labels
SYNOPSIS	<pre>cc [flag ...] file ... -ltsol [library ...] #include <tsol/label.h> int stobcl(const char *string, bclabel_t *label, const int flags, int *error) int stobsl(const char *string, bslabel_t *label, const int flags, int *error) int stobil(const char *string, bilabel_t *label, const int flags, int *error) int stobclear(const char *string, bclear_t *clearance, const int flags, int *error)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to perform label translation on character-coded labels that dominate the process' <i>Sensitivity Label</i>. The stobl functions translate character-coded labels into binary labels. They also modify an existing binary label by incrementing or decrementing it to produce a new binary label relative to its existing value. The generic form of an input character-coded label <i>string</i> is: <pre>[+] [classification name] [[+ -] word ...]</pre> Leading and trailing white space is ignored. Fields are separated by white space, a '/' (slash), or a ',' (comma). Case is irrelevant. If <i>string</i> starts with + or –, <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a <i>classification name</i> followed by a + or –, the new classification is used and the rest of the old <i>label</i> is retained and modified as specified by <i>string</i>. + modifies an existing label by adding <i>Words</i>. – modifies an existing label by removing <i>Words</i>. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>. The stobl functions also translate hexadecimal label representations into binary labels (see hextob()) when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>. <i>flags</i> may be the following: NEW_LABEL <i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be <i>Admin Low</i> for modification changes. If NEW_LABEL is not present, <i>label</i> is validated as a defined label of the correct type dominated by the process' <i>Sensitivity Label</i>. NO_CORRECTION No corrections are made if there are errors in the character-coded label <i>string</i>. <i>string</i> must be complete and contain all the label components that are required by the label_encodings file. The NO_CORRECTION flag implies the NEW_LABEL flag.

ONLY_INFORMATION_LABEL

translate just the *Information Label* portion. (**stobl()** only.)

0 (zero)

The default action is taken.

error is a return parameter that is set only if the function is unsuccessful.

stobl() translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. Where *string* has the form:

[*information label*] [[*sensitivity label*]]

or

‘*’ (star)

flags is the logical sum of **NEW_LABEL** or **NO_CORRECTION** and

ONLY_INFORMATION_LABEL, or is 0 (zero). If both **NEW_LABEL** or **NO_CORRECTION** and **ONLY_INFORMATION_LABEL** are specified, the *Sensitivity Label* portion of *label* is set to the caller's present *Sensitivity Label*. The *sensitivity label* is translated first (unless **ONLY_INFORMATION_LABEL** is specified). Its presence is noted by the [character in the *string*. This translation must result in a *Sensitivity Label* that is dominated by the process' *Sensitivity Label* or an error is reported at the *sensitivity label*. The translated *Sensitivity Label*, or the one present in the *label* parameter must dominate the *information label* that is translated or an error is reported at the *information label*. Unless **NO_CORRECTION** is specified, these translations force the labels to dominate the minimum *Classification*, and initial *Compartments Set* (and *Markings Set*) specified in the **label_encodings** file and correct the label to include other label components that are required by the **label_encodings** file, but not present in *string*. The special case where *string* contains ‘*’ (star) sets the *Sensitivity Label* portion of *label* to the *Information Level* of the *Information Label* portion of *label*.

stobsi() translates the character-coded Sensitivity Label string into a binary sensitivity label and places the result in the return parameter *label*. Where *string* has the form:

[[] *sensitivity label* []]

flags may be either **NEW_LABEL**, **NO_CORRECTION**, or 0 (zero). Unless

NO_CORRECTION is specified, this translation forces the label to dominate the minimum *Classification*, and initial *Compartments Set* specified in the **label_encodings** file and corrects the label to include other label components required by the **label_encodings** file, but not present in *string*.

stobil() translates the character-coded Information Label string into a binary information label and places the result in the return parameter *label*. Where *string* has the form:

information label

flags may be either **NEW_LABEL**, **NO_CORRECTION**, or 0 (zero). Unless

NO_CORRECTION is specified, this translation forces the label to dominate the minimum *Classification*, and initial *Compartments* and *Markings Sets* specified in the **label_encodings** file and corrects the label to include other label components required by the **label_encodings** file, but not present in *string*.

stobclear() translates the character-coded Clearance string into a binary clearance and places the result in the return parameter *Clearance*. Where *string* has the form:

clearance

flags may be either **NEW_LABEL**, **NO_CORRECTION**, or 0 (zero). Unless **NO_CORRECTION** is specified, this translation forces the label to dominate the minimum *Classification*, and initial *Compartments Set* specified in the **label_encodings** file and corrects the label to include other label components that are required by the **label_encodings** file, but not present in *string*. The translation of a *Clearance* may not be the same as the translation of a *Sensitivity Label*. These functions use different tables of the **label_encodings** file that may contain different *Words* and constraints.

RETURN VALUES

These functions return:

- 1** if the translation was successful and a valid binary label was returned.
- 0** if an error occurred. **error** indicates the type of error.

ERRORS

If these functions returned zero, **error** contains one of these values:

- 1** unable to access the **label_encodings** file.
- 0** the label *label* is not valid for this translation and the **NEW_LABEL** or **NO_CORRECTION** flag was not specified, or the label *label* is not dominated by the process' *Sensitivity Label* and the process does not have **PRIV_SYS_TRANS_LABEL** in its set of effective privileges.
- >0** the character-coded label *string* is in error. *error* is a one based index into *string* indicating where the translation error occurred.

FILES

/etc/security/tsol/label_encodings

The label encodings file containing the *Classification* names, *Words*, constraints, and values for the defined labels of this system.

SEE ALSO

bcltobanner(3TSOL), **bilconjoin(3TSOL)**, **blcompare(3TSOL)**, **blinset(3TSOL)**, **blmanifest(3TSOL)**, **blminmax(3TSOL)**, **blportion(3TSOL)**, **bltocolor(3TSOL)**, **bltos(3TSOL)**, **bltype(3TSOL)**, **blvalid(3TSOL)**, **btohex(3TSOL)**, **hextob(3TSOL)**, **labelinfo(3TSOL)**, **labelvers(3TSOL)**, **sbltos(3TSOL)**, **label_encodings(4TSOL)**

Trusted Solaris Developer's Guide

Trusted Solaris user's document set

Trusted Solaris administrator's document set

NOTES

In addition to the *admin low name* and *admin high name* strings defined in the **label_encodings** file, the strings "ADMIN_LOW" and "ADMIN_HIGH" are always accepted as character-coded labels to be translated to the appropriate *Admin Low* and *Admin High* label, respectively.

Modifying an existing *Admin Low* label acts as the specification of a **NEW_LABEL** and forces the label to start at the minimum label specified in the **label_encodings** file.

Modifying an existing *Admin High* label is treated as an attempt to change a label that represents the highest defined *Classification* and all the defined *Compartments* (and, if applicable, *Markings*) specified in the **label_encodings** file.

The **NO_CORRECTION** flag is used when the character-coded label must be complete and accurate so that translation to and from the binary form results in an equivalent character-coded label.

NAME	t6alloc_blk, t6free_blk – Allocate and free security-attribute control structure and buffer
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> t6attr_t t6alloc_blk(t6mask_t mask) void t6free_blk(t6attr_t t6ctl)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Safe
DESCRIPTION	t6alloc_blk allocates a t6attr_t structure, which is an opaque handle used to access the full set of security attributes supported on the system. See man pages for t6get_attr(3NTSOL) and t6set_attr(3NTSOL) for more information on how generic TSIX application programs can access security attributes referenced by t6attr_t without knowing how references are built between the t6attr_t structure and the sets of security attributes for each individual TSIX operating system vendor. If t6alloc_blk is successful, the opaque handle is returned that can be used to set or get individual attributes to or from this control structure. t6alloc_blk allocates space in the control structure for all attributes specified in the <i>mask</i> parameter. t6free_blk should be used in conjunction with t6alloc_blk to free the opaque control structure and any space within it.
RETURN VALUE	Upon successful completion, t6alloc_blk returns a pointer to the t6attr_t structure. Upon failure, t6alloc_blk returns a NULL pointer.
SEE ALSO	libt6(3NTSOL) , t6get_attr(3NTSOL) , t6set_attr(3NTSOL)
WARNING	For generic TSIX applications, use t6free_blk to free memory allocated by t6alloc_blk for better portability.
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6attr_query – Get mask indicating which attributes came from templates
SYNOPSIS	<pre>cc [<i>flags ...</i>] <i>file ...</i> -lt6 #include <tsix/t6attrs.h> int t6attr_query(int <i>fd</i>, t6mask_t *<i>mask</i>)</pre>
DESCRIPTION	Not all security attributes are transmitted with the data. Missing security attributes are taken from a database on the receiving machine. t6attr_query() allows a process to determine when a security attribute comes from a database. A bit value of 1 in the mask indicates that the attribute comes from a database.
RETURN VALUES	Upon successful completion, the call returns 0 . If an error is encountered, it returns -1 .
NOTES	This interface is specific to Trusted Solaris.

NAME	t6clear_blk – Clear security attributes
SYNOPSIS	cc [<i>flags ...</i>] <i>file ...</i> -lt6 #include <tsix/t6attrs.h> void t6clear_blk(t6mask_t mask1, t6attr_t src)
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	t6clear_blk clears attributes specified in <i>mask</i> from the t6attr_t control structure <i>src</i> , which is passed in as an argument.
SEE ALSO	libt6 (3NTSOL), t6alloc_blk (3NTSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6cmp_blk – Compare security attributes
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> int t6cmp_blk(const t6attr_t ctl1, const t6attr_t ctl2)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	t6cmp_blk compares two t6attr_t control structures for the attributes contained. The two t6attr_t control structures are regarded as equal if each type of attribute that exists in <i>ctl1</i> exists in <i>ctl2</i> and vice versa, and if the attribute values of each type in <i>ctl1</i> and <i>ctl2</i> are the same.
RETURN VALUE	This call returns zero if <i>ctl1</i> equals <i>ctl2</i> ; otherwise, the call returns a nonzero value.
SEE ALSO	libt6 (3NTSOL), t6alloc_blk (3NTSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6copy_blk – Copy security attributes
SYNOPSIS	cc [<i>flags ...</i>] <i>file ...</i> -lt6 #include <tsix/t6attrs.h> void t6copy_blk(const t6attr_t attr1, t6attr_t attr2)
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	t6copy_blk copies a set of attributes specified by <i>attr1</i> into the buffers controlled by <i>attr2</i> after both <i>attr1</i> and <i>attr2</i> have been allocated by t6alloc_blk . See man pages for t6alloc_blk (3NTSOL) for more details.
SEE ALSO	libt6 (3NTSOL), t6alloc_blk (3NTSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6dup_blk – duplicate security attributes
SYNOPSIS	<pre>cc [<i>flags</i> ...] <i>file</i> ... -lt6 #include <tsix/t6attrs.h> t6attr_t t6dup_blk(const t6attr_t src)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	t6dup_blk allocates a new t6attr_t control structure and buffer space large enough to hold the set of security attributes in the t6attr_t control structure <i>src</i>, which is passed in as an argument. t6dup_blk then copies that set of attributes specified by <i>src</i> into the newly allocated structure. Upon successful completion, the newly created t6attr_t handle is returned.
RETURN VALUES	Upon successful completion, t6attr_t returns a new handle. If it is unable to allocate sufficient memory for the new attributes, t6dup returns NULL and sets errno to an appropriate value.
ERROR	[ENOMEM] Out of memory for allocation
SEE ALSO	libt6(3NTSOL) , t6alloc_blk(3NTSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6ext_attr, t6new_attr – Manipulate network-endpoint security options
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> int t6ext_attr(int fd, t6cmd_t cmd); int t6new_attr(int fd, t6cmd_t cmd);</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Safe
DESCRIPTION	t6ext_attr turns on extended security operations on the trusted IPC mechanism. <i>fd</i> is the descriptor associated with the IPC mechanism. <i>cmd</i> must be either ON to turn on extended operations or OFF to turn them off. When first created, the trusted IPC mechanism appears the same as an untrusted IPC mechanism. The trusted mechanism can be used in the same way to send and receive data as long as communications do not violate the security policies of the system. Between systems that support mandatory access control, for example, communications can occur only between processes at the same sensitivity level. Before it allows a process to specify security attributes or manipulate the endpoint's security options, the network endpoint must call t6ext_attr. t6new_attr with a cmd value of ON tells the underlying TSIX software that the receiving process is interested in security attributes only if they differ from the last set of attributes received. After this call, t6recvfrom(3NTSOL) returns valid security attributes only when a change in the attributes is detected. This situation is indicated by setting the t6recvfrom parameter <i>new_attrs</i> to nonzero. When new attributes are returned, the full set of requested attributes is returned, not just those that have changed. When <i>cmd</i> is OFF, the default situation prevails: attributes are returned with each call to t6recvfrom.
SEE ALSO	libt6 (3NSTOL), t6recvfrom (3NTSOL)
NOTES	In Trusted Solaris, t6ext_attr is a NULL function. This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6get_attr, t6set_attr – Get security attributes from or set security attributes in the security-attribute buffer handled by a control structure																				
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> void *t6get_attr(t6attr_id_t attr_type, const t6attr_t t6ctl) int t6set_attr(t6attr_id_t attr_type, const void *attr_buf, t6attr_t t6ctl)</pre>																				
AVAILABILITY	SUNWtsolu																				
MT-LEVEL	Safe																				
DESCRIPTION	t6get_attr takes a control structure, <i>t6ctl</i>, and attribute type, <i>attr_type</i>, and returns a pointer to the requested attribute value (type) from the opaque control structure t6ctl. <i>attr_type</i> contains a number (defined in <tsix/t6attrs.h>) that specifies which type of attribute the caller is interested in getting. Only one type can be specified per call. Returned value by t6get_attr should be type cast to the standard type that represents the type indicated by attr_type. t6set_attr replaces the requested attribute value (type) in <i>t6ctl</i> with the value to which <i>attr_buf</i> points. The type of the attribute is specified in <i>attr_type</i> as one of the numbers defined in <tsix/t6attrs.h>.																				
RETURN VALUES	Upon successful completion, t6get_attr returns a pointer to the appropriate value if it exists in the attribute structure. Upon failure, t6get_attr returns NULL. t6set_attr returns 0 if the attribute structure can contain the requested attribute; if not, t6set_attr returns –1 and does not change the attribute structure.																				
SEE ALSO	libt6(3NTSOL) , t6alloc_blk(3NTSOL) , t6free_blk(3NTSOL)																				
NOTES	In Trusted Solaris, t6get_attr returns values of these types: <table> <tr> <td>au_id_t</td><td>Audit ID</td></tr> <tr> <td>auditinfo_t</td><td>Audit info</td></tr> <tr> <td>bclear_t</td><td>Clearance</td></tr> <tr> <td>bilabel_t</td><td>Information label</td></tr> <tr> <td>blabel_t</td><td>Sensitivity label</td></tr> <tr> <td>gid_t</td><td>Effective group ID</td></tr> <tr> <td>gid_t</td><td>Supplemental group IDs</td></tr> <tr> <td>pattr_t</td><td>Process attributes</td></tr> <tr> <td>priv_set_t</td><td>Effective privileges</td></tr> <tr> <td>sid_t</td><td>Session ID</td></tr> </table>	au_id_t	Audit ID	auditinfo_t	Audit info	bclear_t	Clearance	bilabel_t	Information label	blabel_t	Sensitivity label	gid_t	Effective group ID	gid_t	Supplemental group IDs	pattr_t	Process attributes	priv_set_t	Effective privileges	sid_t	Session ID
au_id_t	Audit ID																				
auditinfo_t	Audit info																				
bclear_t	Clearance																				
bilabel_t	Information label																				
blabel_t	Sensitivity label																				
gid_t	Effective group ID																				
gid_t	Supplemental group IDs																				
pattr_t	Process attributes																				
priv_set_t	Effective privileges																				
sid_t	Session ID																				

pid_t	Process ID
uid_t	Effective user ID

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6get_endpt_mask, t6set_endpt_mask, t6get_endpt_default, t6set_endpt_default – Get and set endpoint mask, or get and set endpoint default attributes.
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> int t6get_endpt_mask(int fd, t6mask_t *mask) int t6set_endpt_mask(int fd, t6mask_t mask) int t6get_endpt_default(int fd, t6mask_t *mask, t6attr_t attr_ptr) int t6set_endpt_default(int fd, t6mask_t mask, const t6attr_t attr_ptr)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Safe
DESCRIPTION	The security extensions on the communication endpoint include a set of default security attributes that may be applied to outgoing data and an attribute mask that designates which attributes are taken from the endpoint's default attributes and which are taken from the process's effective attributes. By default, data written to an endpoint has associated with it the security attributes of the process that wrote the data. However, a privileged process may change the value of the default attribute mask on an endpoint the process had created, and the endpoint's default attributes. t6get_endpt_mask allows a process to obtain the current setting of the default attribute mask for the endpoint specified by <i>fd</i>. The attribute mask is returned in the parameter <i>mask</i>. t6set_endpt_mask allows a process to set the bit values of the default attribute mask for the endpoint specified by <i>fd</i> to the value specified by <i>mask</i>. A bit value of 0 indicates the attribute is taken from the process's effective attributes; and a bit value of 1 indicates the attribute is taken from the endpoint's default attributes. t6get_endpt_default allows a process to get the current setting of the default attributes of the endpoint specified by <i>fd</i>. <i>mask</i> indicates which attributes are present in the <i>attr_ptr</i> parameter. To access <i>attr_ptr</i>, see t6get_attr(3NTSOL). t6set_endpt_default allows a process to set the default attributes of the endpoint specified by <i>fd</i> to the attributes specified by <i>attr_ptr</i>. <i>mask</i> indicates which attributes are present in <i>attr_ptr</i>. To set up <i>attr_ptr</i>, see t6set_attr(3NTSOL). Only a process with the appropriate override privileges can change the endpoint's attribute mask or default attributes. To change an endpoint's default attribute or its mask bit, a process must have the override privilege corresponding to the attribute. The override privilege required to specify a default attribute is implementation specific.

RETURN VALUES

Upon successful completion, these calls return **0**. If either call encounters an error, the call returns **-1**.

SEE ALSO

libt6(3NTSOL), **t6sendto(3NTSOL)**, **t6set_attr(3NTSOL)**

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6peek_attr, t6last_attr – Examine the security attributes on the next or the previous byte of data
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> int t6peek_attr(int fd, t6attr_t attr_ptr, t6mask_t *new_attrs) int t6last_attr(int fd, t6attr_t attr_ptr, t6mask_t *new_attrs)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	MT-Safe
DESCRIPTION	t6peek_attr allows a process to peek ahead at the security attributes of the next byte of data. <i>fd</i> is the descriptor of the endpoint; <i>attr_ptr</i> specifies a structure in which to store those attributes the caller wishes to retrieve. <i>new_attrs</i> points to a mask that indicates which attributes were actually retrieved on return from t6peek_attr. t6last_attr allows a process to retrieve the attributes of the last byte of data read from the indicated file descriptor. The parameters for t6last_attr are identical to those for the t6peek_attr routine. If no messages are available at the socket, the examining call waits for a message to arrive, unless the socket is non-blocking (see fcntl(2TSOL)), in which case -1 is returned with the external variable errno set to EWOULDBLOCK.
RETURN VALUES	Upon successful completion, these calls return 0, place the retrieved security attributes in the t6attr_t structure, and set <i>*new_attrs</i> to the mask of those attributes actually returned. If either call encounters an error, it returns -1. When they generate errors, t6peek_attr and t6last_attr set errno.
SEE ALSO	libt6(3NTSOL) , fcntl(2TSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6recvfrom – Read security attributes and data from a trusted endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl -lt6 [library ...] #include <tsix/t6attrs.h> int t6recvfrom(int fd, char *buf, int len, int flags, struct sockaddr *from, int *fromlen, t6attr_t attr_ptr, t6mask_t *new_attrs)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Safe
DESCRIPTION	t6recvfrom receives data and its associated security attributes from a communication endpoint. The <i>from</i> and <i>fromlen</i> parameters are used only if you wish to receive the source address for the data. This information may not be applicable for some trusted endpoints. If not used, these fields should be set to 0. If <i>from</i> is not a NULL pointer, the source address of the message is filled in. <i>fromlen</i> is a value-result parameter, initialized to the size of the buffer associated with <i>from</i>, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket from which the message is received. [See socket(3N).] The <i>flags</i> parameter is formed by ORing one or more of these values: MSG_OOB Read any out-of-band data present on the socket rather than the regular in-band data. If <i>attr_ptr</i> is not NULL, out-of-band data security attributes are also retrieved. MSG_PEEK Peek at the data present on the socket; the data is returned but not consumed, so that a subsequent receive operation will see the same data. If <i>attr_ptr</i> is not NULL, security attributes of the data are also peeked. <i>attr_ptr</i> specifies a control structure in which to store those attributes the caller wishes to retrieve. To get an attribute from the control structure, see t6get_attr(3NTSOL). Any attribute that the receiving process does not care to receive may not be specified in the control structure. This selectivity minimizes the attribute-translation time when passing the attributes out of the kernel. If the t6new_attr(3NTSOL) call was made previously with a setting of ON, the security attributes of the received data will be returned only if they have changed from the last set read. <i>*new_attrs</i> is set to the mask of those attributes actually returned. If new attributes are detected, all attributes requested by the receiving process are returned, not just those that have changed.
RETURN VALUES	Upon success, t6recvfrom returns the number of bytes read. Upon failure, t6recvfrom returns -1 and sets errno .

ERRORS

Always checking the return value is critical. Revocation of access is possible if the received data changes to a level not accessible to the receiving process.

The calls fail if any of these conditions is true:

- EBADF** *fd* is an invalid file descriptor.
- EINTR** The operation was interrupted by delivery of a signal before any data was available to be received.
- EIO** An I/O error occurred while reading from or writing to the file system.
- ENOMEM** There was insufficient user memory available for the operation to complete.
- ENOSR** There were insufficient STREAMS resources available for the operation to complete.
- ENOTSOCK** *fd* is not a socket.
- ESTALE** A stale NFS file handle exists.

SEE ALSO

libt6(3NTSOL), **t6get_attr**(3NTSOL), **t6sendto**(3NTSOL)

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

Only **SOCK_STREAM** sockets created in the **AF_INET** address family support out-of-band data.

NAME	t6sendto – Specify security attributes to send with data on a trusted endpoint				
SYNOPSIS	<pre>cc [flag ...] file ... -lsocket -lnsl -lt6 [library ...] #include <tsix/t6attrs.h> int t6sendto(int fd, const char *msg, int len, int flags, const struct sockaddr *to, int tolen, const t6attr_t attr_ptr)</pre>				
AVAILABILITY	SUNWtsolu				
MT-LEVEL	Safe				
DESCRIPTION	t6sendto allows a privileged process to specify the security attributes to send with an IPC message. A process may specify only those attributes for which it possesses the appropriate override privilege and need not specify a full set. Any unspecified attributes are supplied by the kernel. <i>fd</i> is a socket created with socket(3N). The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i>. The <i>to</i> pointer and <i>to_len</i> parameter are used only if you are specifying the destination address; otherwise they should be set to 0. You may not specify the address if the trusted endpoint was created for a connection-oriented protocol, such as TCP. If the message is too long to pass atomically through the underlying protocol, then the message is not transmitted and the error EMSGSIZE is returned. A return value of -1 indicates locally detected errors only, not implicitly that the message was not delivered. The <i>flags</i> parameter is formed from the bitwise OR of zero or more of these values: <table> <tr> <td>MSG_OOB</td><td>Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.</td></tr> <tr> <td>MSG_DONTROUTE</td><td>The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.</td></tr> </table> The security attributes are specified by the <i>attr_ptr</i> parameter. To set up <i>attr_ptr</i>, see t6set_attr(3NTSOL). Only a process with the appropriate override privileges can specify the security attributes associated with the data it sends. To specify an attribute, a process must have the override privilege corresponding to the attribute. The override privilege required to specify an attribute is implementation specific. For Trusted Solaris, one or more of these privileges may be required: PRIV_NET_DOWNGRADE_SL, PRIV_NET_UPGRADE_SL, PRIV_NET_DOWNGRADE_IL, PRIV_NET_UPGRADE_IL, PRIV_NET_SETCLR,	MSG_OOB	Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.
MSG_OOB	Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.				

PRIV_NET_SETID, PRIV_NET_SETPRIV, PRIV_NET_BROADCAST.

RETURN VALUES

Upon success, the return value is the number of bytes actually sent. Upon failure, the call returns **-1** and sets the error code in **errno**.

Always checking the return value is critical, for the addition of security means that access to an endpoint may be revoked in response to a security violation.

ERRORS

t6sendto fails if any of these conditions is true:

- EBADF** *fd* is an invalid file descriptor.
- EINTR** The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
- EINVAL** *tolen* is not the size of a valid address for the specified address family.
- EMSGSIZE** The socket requires that message be sent atomically, and the message was too long.
- ENOMEM** There was insufficient memory available to complete the operation.
- ENOSR** There were insufficient STREAMS resources available for the operation to complete.
- ENOTSOCK** *fd* is not a socket.

SEE ALSO

libt6(3NTSOL), **t6set_attr(3NTSOL)**, **t6set_endpt_default(3NTSOL)**, *Trusted Solaris Developer's Guide*

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

Only **SOCK_STREAM** sockets created in the **AF_INET** address family support out-of-band data.

NAME	t6size_attr - Get the size of a particular attribute from the control structure
SYNOPSIS	<pre>cc [flags ...] file ... -lt6 #include <tsix/t6attrs.h> int t6size_attr(t6attr_id_t attr_type, const t6attr_t t6ctl)</pre>
AVAILABILITY	SUNWtsolu
MT-LEVEL	Safe
DESCRIPTION	t6size_attr returns the size of an attribute indicated by <i>attr_type</i>. If the t6attr_t control structure <i>t6ctl</i> is a NULL pointer, t6size_attr returns either the size of a fixed-size attribute or the maximum size of a variable-size attribute. If the <i>attr_type</i> is invalid, t6size_attr returns 0. If the t6attr_t control structure <i>t6ctl</i> is not NULL, t6size_attr returns either the size of a fixed-size attribute or the actual size occupied by a variable-size attribute in the control structure <i>t6ctl</i>. If the <i>attr_type</i> is invalid or not in the <i>t6ctl</i>, t6size_attr returns 0.
SEE ALSO	t6dup_attr (3NTSOL), t6copy_attr (3NTSOL)
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t_accept – Accept a connect request
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_accept(int <i>fildev</i>, int <i>resfd</i>, struct t_call *<i>call</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	This function is issued by a transport user to accept a connect request. <i>fildev</i> identifies the local-transport endpoint where the connect indication arrived, <i>resfd</i> specifies the local-transport endpoint where the connection is to be established, and <i>call</i> contains information required by the transport provider to complete the connection. <i>call</i> points to a t_call structure that contains these members: <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; </pre> struct netbuf contains these members: <pre> unsigned int maxlen; unsigned int len; char * buf; </pre> In <i>call</i>, addr is the address of the caller, opt indicates any protocol-specific parameters associated with the connection, udata points to any user data to be returned to the caller, and sequence is the value returned by t_listen that uniquely associates the response with a previously received connect indication. A transport user may accept a connection either on the same or on a different, local-transport endpoint from the one on which the connect indication arrived. If the same endpoint is specified (that is, <i>resfd</i>=<i>fildev</i>), the connection can be accepted unless the user has received other indications on that endpoint but has not responded to them [with t_accept() or t_snddis(3N)]. For this condition, t_accept will fail and set t_errno to TBADF. If a different transport endpoint is specified (<i>resfd</i>!=<i>fildev</i>), the endpoint must be bound to a protocol address and must be in the T_IDLE state [see t_getstate(3N)] before the t_accept is issued. For both types of endpoints, t_accept will fail and set t_errno to TLOOK if there are indications (for example, a connect or disconnect) waiting to be received on that endpoint. The values of parameters specified by opt and the syntax of those values are protocol specific. The udata argument enables the called transport user to send user data to the caller; the amount of user data must not exceed the limits supported by the transport provider as returned in the connect field of the <i>info</i> argument of t_open(3N) or t_getinfo(3N). If the len field of udata is zero, no data will be sent to the caller.

RETURN VALUES	Upon successful completion, t_accept returns a value of 0 . Otherwise, t_accept returns a value of -1 , sets t_errno to indicate the error, and possibly sets errno .
ERRORS	Upon failure, t_errno will be set to one of these vaues: TACCES The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options. TBADF The specified file descriptor does not refer to a transport endpoint, or the user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived. TBADDATA The amount of user data specified was not within the bounds allowed by the transport provider. TBADOPT The specified options were in an incorrect format or contained illegal information. TBADSEQ An invalid sequence number was specified. TLOOK An asynchronous event has occurred on the transport endpoint referenced by <i>fildev</i> and requires immediate attention. TNOTSUPPORT This function is not supported by the underlying transport provider. TOUTSTATE The function was issued in the wrong sequence on the transport endpoint referenced by <i>fildev</i>, or the transport endpoint to which <i>resfd</i> refers is not in the T_IDLE state. TSYSERR A system error has occurred during execution of this function; errno will be set to the specific error.
SUMMARY OF TRUSTED SOLARIS CHANGES	If the calling process possesses the PRIV_NET_MAC_READ privilege and the socket has been bound to a multilevel port (MLP), the connection is accepted on a MLP; otherwise, the connection is accepted on a single-level port (SLP). [See bind(3NTSOL) for more information.]
SEE ALSO	t_accept(3N), t_connect(3N), t_getinfo(3N), t_getstate(3N), t_listen(3N), t_open(3N), t_snddis(3N), t_rcvconnect(3N) <i>Transport Interfaces Programming Guide</i>
NOTES	This interface is safe in multithreaded applications.

NAME	t_bind – Bind an address to a transport endpoint
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_bind(int fildes, const struct t_bind *req, struct t_bind *ret);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	This function associates a protocol address with the transport endpoint specified by <i>fildes</i> and activates that transport endpoint. In connection mode, the transport provider may begin accepting or requesting connections on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint. The <i>req</i> and <i>ret</i> arguments point to a t_bind structure containing these members: <pre>struct netbuf addr; unsigned qlen;</pre> netbuf is described in t_connect(3N). The addr field of the t_bind structure specifies a protocol address and the qlen field is used to indicate the maximum number of outstanding connect indications. <i>req</i> is used to request that an address, represented by the netbuf structure, be bound to the given transport endpoint. len [see netbuf in t_connect(3N) for len, buf, and maxlen] specifies the number of bytes in the address and buf points to the address buffer. maxlen has no meaning for the <i>req</i> argument. On return, <i>ret</i> contains the address that the transport provider actually bound to the transport endpoint; this address may be different from the address specified by the user in <i>req</i>. In <i>ret</i>, the user specifies maxlen, which is the maximum size of the address buffer, and buf, which points to the buffer where the address is to be placed. On return, len specifies the number of bytes in the bound address and buf points to the bound address. If maxlen is not large enough to hold the returned address, an error will result. If the requested address is not available or if no address is specified in <i>req</i> (the len field of addr in <i>req</i> is zero), the transport provider may assign an appropriate address to be bound, and will return that address in the addr field of <i>ret</i>. The user can compare the addresses in <i>req</i> and <i>ret</i> to determine whether the transport provider bound the transport endpoint to a different address from that requested. <i>req</i> may be NULL if the user does not wish to specify an address to be bound. Here, the value of qlen is assumed to be zero, and the transport provider must assign an address to the transport endpoint. Similarly, <i>ret</i> may be NULL if the user does not care what address was bound by the provider and is not interested in the negotiated value of qlen. It is valid to set <i>req</i> and <i>ret</i> to NULL for the same call; in that case, the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

NAME	t_optmgmt – manage options for a transport endpoint						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...]</pre> <pre>#include <tiuser.h></pre> <pre>int t_optmgmt(int <i>filides</i>, const struct t_optmgmt *<i>req</i>, struct t_optmgmt *<i>ret</i>);</pre>						
MT-LEVEL	MT-Safe						
DESCRIPTION	The t_optmgmt() function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. <i>filides</i> identifies a bound transport endpoint. The <i>req</i> and <i>ret</i> arguments point to a t_optmgmt structure containing the following members: <pre> struct netbuf opt; long flags;</pre> The opt field identifies protocol options and the flags field is used to specify the action to take with those options. The options are represented by a netbuf (see t_connect(3N); also for len, buf, and maxlen) structure in a manner similar to the address in t_bind(3N). <i>req</i> is used to request a specific action of the provider and to send options to the provider. len specifies the number of bytes in the options, buf points to the options buffer, and maxlen has no meaning for the <i>req</i> argument. The transport provider may return options and flag values to the user through <i>ret</i>. For <i>ret</i>, maxlen specifies the maximum size of the options buffer and buf points to the buffer where the options are to be placed. On return, len specifies the number of bytes of options returned. maxlen has no meaning for the <i>req</i> argument, but must be set in the <i>ret</i> argument to specify the maximum number of bytes the options buffer can hold. The actual structure and content of the options is imposed by the transport provider. The flags field of <i>req</i> can specify one of the following actions: <table> <tr> <td>T_NEGOTIATE</td><td>This action enables the user to negotiate the values of the options specified in <i>req</i> with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through <i>ret</i>.</td></tr> <tr> <td>T_CHECK</td><td>This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the flags field of <i>ret</i> will have either T_SUCCESS or T_FAILURE set to indicate to the user whether the options are supported. These flags are only meaningful for the T_CHECK request.</td></tr> <tr> <td>T_DEFAULT</td><td>This action enables a user to retrieve the default options supported by the transport provider into the opt field of <i>ret</i>. In <i>req</i>, the len field of opt must be zero and the buf field may be NULL.</td></tr> </table>	T_NEGOTIATE	This action enables the user to negotiate the values of the options specified in <i>req</i> with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through <i>ret</i> .	T_CHECK	This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the flags field of <i>ret</i> will have either T_SUCCESS or T_FAILURE set to indicate to the user whether the options are supported. These flags are only meaningful for the T_CHECK request.	T_DEFAULT	This action enables a user to retrieve the default options supported by the transport provider into the opt field of <i>ret</i> . In <i>req</i> , the len field of opt must be zero and the buf field may be NULL.
T_NEGOTIATE	This action enables the user to negotiate the values of the options specified in <i>req</i> with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through <i>ret</i> .						
T_CHECK	This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the flags field of <i>ret</i> will have either T_SUCCESS or T_FAILURE set to indicate to the user whether the options are supported. These flags are only meaningful for the T_CHECK request.						
T_DEFAULT	This action enables a user to retrieve the default options supported by the transport provider into the opt field of <i>ret</i> . In <i>req</i> , the len field of opt must be zero and the buf field may be NULL.						

	If issued as part of the connectionless-mode service, t_optmgmt() may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.														
SUMMARY OF TRUSTED SOLARIS CHANGES	A process must have the PRIV_NET_RAWACCESS privilege in order to specify IP options 130 or 134. The former refers to the RIPS0 Basic Security Option and the later refers to the CIPSO option.														
RETURN VALUES	t_optmgmt() returns 0 on success. On failure t_optmgmt() returns -1, t_errno is set to indicate the error, and possibly errno is set.														
ERRORS	On failure, t_errno will be set to one of the following: <table> <tr> <td>TBADF</td><td>The specified file descriptor does not refer to a transport endpoint.</td></tr> <tr> <td>TACCES</td><td>The user does not have permission to negotiate the specified options.</td></tr> <tr> <td>TBADFLAG</td><td>An invalid flag was specified.</td></tr> <tr> <td>TBADOPT</td><td>The specified protocol options were in an incorrect format or contained illegal information.</td></tr> <tr> <td>TBUFOVFLW</td><td>The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in <i>ret</i> will be discarded.</td></tr> <tr> <td>TOUTSTATE</td><td>The function was issued in the wrong sequence.</td></tr> <tr> <td>TSYSERR</td><td>A system error has occurred during execution of this function, or the specified option requires PRIV_NET_RAWACCESS privilege. errno will be set to the specific error.</td></tr> </table>	TBADF	The specified file descriptor does not refer to a transport endpoint.	TACCES	The user does not have permission to negotiate the specified options.	TBADFLAG	An invalid flag was specified.	TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.	TBUFOVFLW	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in <i>ret</i> will be discarded.	TOUTSTATE	The function was issued in the wrong sequence.	TSYSERR	A system error has occurred during execution of this function, or the specified option requires PRIV_NET_RAWACCESS privilege. errno will be set to the specific error.
TBADF	The specified file descriptor does not refer to a transport endpoint.														
TACCES	The user does not have permission to negotiate the specified options.														
TBADFLAG	An invalid flag was specified.														
TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.														
TBUFOVFLW	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in <i>ret</i> will be discarded.														
TOUTSTATE	The function was issued in the wrong sequence.														
TSYSERR	A system error has occurred during execution of this function, or the specified option requires PRIV_NET_RAWACCESS privilege. errno will be set to the specific error.														
SEE ALSO	t_bind(3N), t_connect(3N), t_getinfo(3N), t_open(3N) <i>Transport Interfaces Programming Guide</i>														
NOTES	This interface is safe in multithreaded applications.														

NAME	t_snd – Send data or expedited data over a connection
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <tiuser.h> int t_snd(int <i>fildev</i>, char *<i>buf</i>, unsigned <i>nbytes</i>, int <i>flags</i>);</pre>
MT-LEVEL	MT-Safe
DESCRIPTION	This function is used to send either normal or expedited data. <i>fildev</i> identifies the local-transport endpoint over which data should be sent, <i>buf</i> points to the user data, <i>nbytes</i> specifies the number of bytes of user data to be sent, and <i>flags</i> specifies any optional flags described later. By default, t_snd() operates in synchronous mode and may wait if flow-control restrictions prevent the data from being accepted by the local-transport provider at the time the call is made. However, if O_NDELAY or O_NONBLOCK is set [using t_open(3N) or fcntl(2)], t_snd will execute in asynchronous mode and will fail immediately if there are flow-control restrictions. Even when there are no flow-control restrictions, t_snd will wait if STREAMS internal resources are not available, regardless of the state of O_NDELAY or O_NONBLOCK. Upon successful completion, t_snd returns the number of bytes accepted by the transport provider. Normally this number will equal the number of bytes specified in <i>nbytes</i>. However, if O_NDELAY or O_NONBLOCK is set, it is possible that only part of the data will be accepted by the transport provider. In this case, t_snd will set T_MORE for the data that was sent (see later) and will return a value less than <i>nbytes</i>. If <i>nbytes</i> is zero and sending of zero bytes is not supported by the underlying transport provider, t_snd will return -1 with t_errno set to TBADDATA. A return value of zero indicates that the request to send a zero-length data message was sent to the provider. If T_EXPEDITED is set in <i>flags</i>, the data will be sent as expedited data and will be subject to the interpretations of the transport provider. If T_MORE is set in <i>flags</i> or is set as described later, an indication is sent to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple t_snd calls. Each t_snd with the T_MORE flag set indicates that another t_snd will follow with more data for the current TSDU. The end of the TSDU (or ETSDU) is identified by a t_snd call with the T_MORE flag not set. Use of T_MORE enables a user to break up large, logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the <i>info</i> argument on return from t_open(3N) or t_getinfo(3N), the T_MORE flag is not meaningful and should be ignored.

The size of each **TSDU** or **ETSDU** must not exceed the limits of the transport provider as returned by **t_open(3N)** or **t_getinfo(3N)**. If the size is exceeded, a **TSYSERR** with system error **EPROTO** will occur. However, the **t_snd** may not fail because **EPROTO** errors may not be reported immediately. In this case, a subsequent call that accesses the transport endpoint will fail with the associated **TSYSERR**.

If **t_snd** is issued from the **T_IDLE** state, the provider may silently discard the data. If **t_snd** is issued from any state other than **T_DATAXFER**, **T_INREL**, or **T_IDLE**, the provider will generate a **TSYSERR** with system error **EPROTO** (which may be reported in the manner described earlier).

RETURN VALUES

Upon successful completion, **t_snd** returns the number of bytes accepted by the transport provider. Upon failure **t_snd** returns **-1**, sets **t_errno** to indicate the error, and possibly sets **errno**.

ERRORS

Upon failure, **t_errno** will be set to one of these values:

- TBADDATA** *nbytes* is zero and sending zero bytes is not supported by the transport provider.
- TBADF** The specified file descriptor does not refer to a transport endpoint.
- TFLOW** **O_NDELAY** or **O_NONBLOCK** was set, but the flow-control mechanism prevented the transport provider from accepting data at this time.
- TNOTSUPPORT** This function is not supported by the underlying transport provider.
- TSYSERR** A system error [see **intro(2)**] has been detected during execution of this function.

SUMMARY OF TRUSTED SOLARIS CHANGES

If the process calling these routines possesses the **PRIV_NET_REPLY_EQUAL** privilege, the packets the process sends will carry the same CMW label and clearance as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

fcntl(2TSOL), **t_getinfo(3N)**, **t_open(3N)**, **t_rcv(3N)**

Transport Interfaces Programming Guide

NOTES

This interface is safe in multithreaded applications.

NAME	t_sndudata – Send a data unit						
SYNOPSIS	<pre>cc [flag ...] file ... -lnsl [library ...] #include <tiuser.h> int t_sndudata(int fildes, struct t_unitdata *unitdata);</pre>						
MT-LEVEL	MT-Safe						
DESCRIPTION	This function is used in connectionless mode to send a data unit to another transport user. <i>fildes</i> identifies the local-transport endpoint through which data will be sent, and <i>unitdata</i> points to a t_unitdata structure containing these members: <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> netbuf is described in t_connect(3N). In <i>unitdata</i>, addr specifies the protocol address of the destination user, opt identifies protocol-specific options that the user wants associated with this request, and udata specifies the user data to be sent. The user may choose not to specify what protocol options are associated with the transfer by setting the len field of opt to zero. In this case, the provider may use default options. If the len field of udata is zero and the sending of zero bytes is not supported by the underlying transport provider, t_sndudata will return -1 with t_errno set to TBADDATA. By default, t_sndudata() operates in synchronous mode and may wait if flow-control restrictions prevent the data from being accepted by the local-transport provider at the time the call is made. However, if O_NDELAY or O_NONBLOCK is set [using t_open(3N) or fcntl(2)], t_sndudata will execute in asynchronous mode and will fail. If t_sndudata is issued from an invalid state, or if the amount of data specified in udata exceeds the TSDU size as returned in the tsdu field of the <i>info</i> argument of t_open(3N) or t_getinfo(3N), the provider will generate an EPROTO protocol error. [See TSYSERR.) If the state is invalid, this error may not occur until a subsequent reference is made to the transport endpoint.						
RETURN VALUES	Upon successful completion, t_sndudata returns 0 . Upon failure, t_sndudata returns -1 , sets t_errno to indicate the error, and possibly sets errno .						
ERRORS	Upon failure, t_errno will be set to one of these values: <table> <tr> <td>TBADDATA</td><td>nbytes is zero and sending zero bytes is not supported by the transport provider.</td></tr> <tr> <td>TBADF</td><td>The specified file descriptor does not refer to a transport endpoint.</td></tr> <tr> <td>TFLOW]</td><td>O_NDELAY or O_NONBLOCK was set, but the flow-control mechanism prevented the transport provider from accepting data at this time.</td></tr> </table>	TBADDATA	nbytes is zero and sending zero bytes is not supported by the transport provider.	TBADF	The specified file descriptor does not refer to a transport endpoint.	TFLOW]	O_NDELAY or O_NONBLOCK was set, but the flow-control mechanism prevented the transport provider from accepting data at this time.
TBADDATA	nbytes is zero and sending zero bytes is not supported by the transport provider.						
TBADF	The specified file descriptor does not refer to a transport endpoint.						
TFLOW]	O_NDELAY or O_NONBLOCK was set, but the flow-control mechanism prevented the transport provider from accepting data at this time.						

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES****SEE ALSO****NOTES**

TNOTSUPPORT This function is not supported by the underlying transport provider.

TSYSERR A system error has occurred during execution of this function; **errno** will be set to the specific error.

If the process calling these routines possesses the **PRIV_NET_REPLY_EQUAL** privilege, the packets the process sends will carry the same CMW label and clearance as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

fcntl(2TSOL), **t_connect(3N)**, **t_getinfo(3N)**, **t_rcvudata(3N)**, **t_rcvuderr(3N)**

Transport Interfaces Programming Guide

This interface is safe in multithreaded applications.

Index

A

accept — accept a connection on a socket, 3TSOL-49
accept(3TSOL), 3TSOL-7
access utmpx file entry

- **endutxent**, 3CTSOL-122
- **getutmp**, 3CTSOL-122
- **getutmpx**, 3CTSOL-122
- **getutxent**, 3CTSOL-122
- **getutxid**, 3CTSOL-122
- **getutxline**, 3CTSOL-122
- **pututxline**, 3CTSOL-122
- **setutxent**, 3CTSOL-122
- **updwtmp**, 3CTSOL-122
- **updwtmpx**, 3CTSOL-122
- **utmpxname**, 3CTSOL-122

adornfc() — adorn the final component of a path-name, 3TSOL-51
au_class_x_t_alloc — Extended audit class manipulation functions, 3TSOL-52
au_class_x_t_free — Extended audit class manipulation functions, 3TSOL-52
au_preselect — preselect an audit record, 3TSOL-53
au_user_mask — get user's binary preselection mask, 3TSOL-55
audit control file information

- **endac**, 3TSOL-97

audit control file information, *continued*

- **getacdir**, 3TSOL-97
- **getacflg**, 3TSOL-97
- **getacinfo**, 3TSOL-97
- **getacmin**, 3TSOL-97
- **getacna**, 3TSOL-97
- **setac**, 3TSOL-97

audit record tokens, manipulating

- **au_preselect**, 3TSOL-53

auditing functions

- **au_preselect**, 3TSOL-53

auditwrite

- **uditwrite**, 3TSOL-56

auditwrite error messages

- **aw_errno**, 3TSOL-67
- **aw_perror**, 3TSOL-67
- **aw_strerror**, 3TSOL-67

B

Basic Security Module functions

- **au_user_mask**, 3TSOL-55

bclearhigh() — initialize *Admin High Binary Clearance*, 3TSOL-78
bclearlow() — initialize *Admin Low Binary Clearance*, 3TSOL-78
bcleartoh() — *Binary Clearance* to hexadecimal string, 3TSOL-90
bcleartoh_r() — *Binary Clearance* to hexadecimal string, 3TSOL-90

cimal string, 3TSOL-90
 bcleartos() – *Binary Clearance* to string, 3TSOL-85
 bclearundef() – initialize *Undefined Binary Clearance*, 3TSOL-78
 bclearvalid() – check validity of binary clearance, 3TSOL-89
 bclhigh() – initialize *Admin High Binary CMW Label*, 3TSOL-78
 bcllow() – initialize *Admin Low Binary CMW Label*, 3TSOL-78
 bcltobanner() – translate *Binary CMW Label* to printer banner page fields , 3TSOL-68
 bcltoh() – *Binary CMW Label* to hexadecimal string, 3TSOL-90
 bcltoh_r() – *Binary CMW Label* to hexadecimal string, 3TSOL-90
 bcltoil() – reference *Information Label*, 3TSOL-81
 bcltos() – *Binary CMW Label* to string, 3TSOL-85
 bcltosl() – reference *Sensitivity Label*, 3TSOL-81
 bclundef() – initialize *Undefined Binary CMW Label*, 3TSOL-78
 bilconjoin() – conjoin two binary information labels, 3TSOL-71
 bildominates() – compare *Information Labels* for dominance, 3TSOL-74
 bilequal() – compare *Information Labels* for equality, 3TSOL-74
 bilhigh() – initialize *Admin High Binary Information Label*, 3TSOL-78
 billow() – initialize *Admin Low Binary Information Label*, 3TSOL-78
 biltoh() – *Binary Information Label* to hexadecimal string, 3TSOL-90
 biltoh_r() – *Binary Information Label* to hexadecimal string, 3TSOL-90
 biltolev() – reference *Information Level*, 3TSOL-81

biltos() – *Binary Information Label* to string, 3TSOL-85
 bilvalid() – check validity of binary information label, 3TSOL-89
 bilundef() – initialize *Undefined Binary Information Label*, 3TSOL-78
 bimdominates() – compare *Information Label Markings Sets* for dominance, 3TSOL-74
 bimequal() – compare *Information Label Markings Sets* for equality, 3TSOL-74
 bind – bind a name to a socket, 3NTSOL-72
 bind an address to a transport endpoint – t_bind, 3NTSOL-240
 bldominates() – compare levels for dominance, 3TSOL-74
 blequal() – compare levels for equality, 3TSOL-74
 blinrange() – compare level to be between bounding levels, 3TSOL-74
 blinset() – check level for set inclusion, 3TSOL-76
 blmaximum() – least upper bound of two *Binary Levels*, 3TSOL-80
 blminimum() – greatest lower bound two *Binary Levels*, 3TSOL-80
 blstrictdom() – compare levels for strict dominance, 3TSOL-74
 bltocolor() – get ASCII color name of label, 3TSOL-83
 bltocolor_r() – get ASCII color name of label, 3TSOL-83
 bltype() – check type of *Binary Label*, 3TSOL-88
 bslhigh() – initialize *Admin High Binary Sensitivity Label*, 3TSOL-78
 bsllow() – initialize *Admin Low Binary Sensitivity Label*, 3TSOL-78
 bslltoh() – *Binary Sensitivity Label* to hexadecimal string, 3TSOL-90
 bslltoh_r() – *Binary Sensitivity Label* to hexadecimal string, 3TSOL-90

bsltos() – *Binary Sensitivity Label* to string, 3TSOL-85
bslvalid() – check validity of binary sensitivity label, 3TSOL-89
bslundef() – initialize *Undefined Binary Sensitivity Label*, 3TSOL-78

C

clnt_call – library routines for client side calls, 3NTSOL-188
clnt_control – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_create – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_create_timed – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_create_vers – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_destroy – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_dg_create – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_freeres – library routines for client side calls, 3NTSOL-188
clnt_geterr – library routines for client side calls, 3NTSOL-188
clnt_pcreateerror – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_perrno – library routines for client side calls, 3NTSOL-188
clnt_perror – library routines for client side calls, 3NTSOL-188
clnt_raw_create – library routines for

dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_spcreateerror – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_sperrno – library routines for client side calls, 3NTSOL-188
clnt_sperror – library routines for client side calls, 3NTSOL-188
clnt_tli_create – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_tp_create – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_tp_create_timed – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clnt_vc_create – library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192
clock_getres – high-resolution clock operations, 3RTSOL-92
clock_gettime – high-resolution clock operations, 3RTSOL-92
clock_settime – high-resolution clock operations, 3RTSOL-92
communications
 accept a connect request – t_accept, 3NTSOL-238
 accept a connection on a socket – accept, 3NTSOL-49
 bind a name to a socket – bind, 3NTSOL-72
 bind an address to a transport endpoint – t_bind, 3NTSOL-240
 listen for connections on a socket – listen, 3NTSOL-143
 manage options for a transport endpoint – t_optmgmt, 3NTSOL-242
 send a data unit – t_sndudata,

3NTSOL-246
communications, *continued*
 send a message from a socket —
 send, sendto, sendmsg,
 3NTSOL-214
 send data or expedited data over a connection
 — t_snd, 3NTSOL-244
current working directory
 get pathname — **mldmldgetcwd**, 3TSOL-144

D

directories
 get pathname of current working directory —
 mldgetcwd, 3TSOL-144
dn_comp — resolver routines, 3NTSOL-177
dn_expand — resolver routines, 3NTSOL-177

E

endac — get audit control file information,
 3TSOL-97
endauclass — close audit_class database file,
 3TSOL-99
endauevent — close audit_event database file,
 3TSOL-102
endauser — get audit_user database entry,
 3TSOL-104
endutent — access utmp file entry, 3CTSOL-119
endutxent — access utmpx file entry, 3CTSOL-122

F

file tree
 recursively descend — ftw, 3CTSOL-94
ftw — walk a file tree, 3CTSOL-94

G

get
 user audit characteristics for socket or TLI
 peer— getpeerinfo, 3TSOL-108
getacdir — get audit control file information,
 3TSOL-97
getacflg — get audit control file information,
 3TSOL-97
getacinfo — get audit control file information,

3TSOL-97
getacmin — get audit control file information,
 3TSOL-97
getacna — get audit control file information,
 3TSOL-97
getauclassent — get audit_class database entry,
 3TSOL-99
getauclassent_r — get audit_class database
 entry, 3TSOL-99
getauclassnam — get audit_class database entry,
 3TSOL-99
getauclassnam_r — get audit_class database
 entry, 3TSOL-99
getauditflags() — generate process audit state,
 3TSOL-106
getauditflagsbin() — convert audit flag
 specifications, 3TSOL-101
getauditflagschar() — convert audit flag
 specifications, 3TSOL-101
getauevent — get audit_event database entry,
 3TSOL-102
getauevent_r — get audit_event database entry,
 3TSOL-102
getauevnam — get audit_event database entry,
 3TSOL-102
getauevnam_r — get audit_event database entry,
 3TSOL-102
getauevnonam — get audit_event database entry,
 3TSOL-102
getauevnum — get audit_event database entry,
 3TSOL-102
getauevnum_r — get audit_event database entry,
 3TSOL-102
getauuserent — get audit_user database entry,
 3TSOL-104
getauuserent_r — get audit_user database entry,
 3TSOL-104
getauusername — get audit_user database entry,
 3TSOL-104
getauusername_r — get audit_user database entry,
 3TSOL-104
getcil() — extract *Information Label*, 3TSOL-81

`getcs1()` — extract *Sensitivity Label*, 3TSOL-81
`getpeerinfo`, 3TSOL-108
`getutent` — access utmp file entry, 3CTSOL-119
`getutid` — access utmp file entry, 3CTSOL-119
`getutline` — access utmp file entry, 3CTSOL-119
`getutmp` — access utmpx file entry, 3CTSOL-122
`getutmpx` — access utmpx file entry, 3CTSOL-122
`getutxent` — access utmpx file entry, 3CTSOL-122
 `endutxent()`, 3CTSOL-122
 `getutmp()`, 3CTSOL-122
 `getutmpx()`, 3CTSOL-122
 `getutxent()`, 3CTSOL-122
 `getutxid()`, 3CTSOL-122
 `getutxline()`, 3CTSOL-122
 `pututxline()`, 3CTSOL-122
 `setutxent()`, 3CTSOL-122
 `updwtmp()`, 3CTSOL-122
 `updwtmpx()`, 3CTSOL-122
 `utmpxname()`, 3CTSOL-122
`getutxid` — access utmpx file entry, 3CTSOL-122
`getutxline` — access utmpx file entry,
 3CTSOL-122
`getvfsaent` — get vfstab_adjunct file entry,
 3TSOL-125
`getvfsafile` — get vfstab_adjunct file entry,
 3TSOL-125
group IDs, supplementary
 initialize — `initgroups`, 3CTSOL-127

H

`h_alloc()` — Allocate memory for a hexadecimal
 string, 3TSOL-90
`h_free()` — free hexadecimal string memory,
 3TSOL-90
`htobcl()` — hexadecimal string to *Binary CMW*
 Label, 3TSOL-126
`htobclear()` — hexadecimal string to *Binary*
 Clearance, 3TSOL-126
`htobil()` — hexadecimal string to *Binary Informa-*
 tion Label, 3TSOL-126
`htobsl()` — hexadecimal string to *Binary Sensi-*
 tivity Label, 3TSOL-126

I

`initgroups` — initialize the supplementary group
 access list, 3CTSOL-127

K

kernel virtual memory functions
 `kstat_read` — read or write kstat data,
 3KTSOL-128
 `kstat_write` — read or write kstat data,
 3KTSOL-128
`kstat_read` — read or write kstat data,
 3KTSOL-128
`kstat_write` — read or write kstat data,
 3KTSOL-128

L

`tsol_lbuild_create()`
 `tsol_lbuild_create()`, 3TSOL-129
 `tsol_lbuild_destroy()`, 3TSOL-129
 `tsol_lbuild_get()`, 3TSOL-129
 `tsol_lbuild_set()`, 3TSOL-129
`Xbclearertos()`
 `Xbclearertos()`, 3TSOL-134
 `Xbcltos()`, 3TSOL-134
 `Xbiltos()`, 3TSOL-134
 `Xbsltos()`, 3TSOL-134
`bclearhigh()`
 `bclearhigh()`, 3TSOL-78
 `bclearlow()`, 3TSOL-78
 `bcleartoh()`, 3TSOL-90
 `bcleartoh_r()`, 3TSOL-90
 `bclearertos()`, 3TSOL-85
 `bclearundef()`, 3TSOL-78
 `bclearvalid()`, 3TSOL-89
 `bclhigh()`, 3TSOL-78
 `bcllow()`, 3TSOL-78
 `bcltobanner()`, 3TSOL-68
 `bcltoh()`, 3TSOL-90
 `bcltoh_r()`, 3TSOL-90
 `bcltoil()`, 3TSOL-81
 `bcltos()`, 3TSOL-85
 `bcltosl()`, 3TSOL-81
 `bclundef()`, 3TSOL-78
 `bilconjoin()`, 3TSOL-71

bclearhigh(), *continued*
 bldominates(), 3TSOL-74
 bilequal(), 3TSOL-74
 bilow(), 3TSOL-78
 biltoh(), 3TSOL-90
 biltoh_r(), 3TSOL-90
 biltolev(), 3TSOL-81
 biltos(), 3TSOL-85
 bilundef(), 3TSOL-78
 bilvalid(), 3TSOL-89
 bimdominates(), 3TSOL-74
 bimequal(), 3TSOL-74
 bldominates(), 3TSOL-74
 blequal(), 3TSOL-74
 blinrange(), 3TSOL-74
 blinset(), 3TSOL-76
 blmaximum(), 3TSOL-80
 blminimum(), 3TSOL-80
 blstrictdom(), 3TSOL-74
 bltcolor(), 3TSOL-83
 bltcolor_r(), 3TSOL-83
 bltype(), 3TSOL-88
 bslhigh(), 3TSOL-78
 bsllow(), 3TSOL-78
 bsltoh(), 3TSOL-90
 bsltoh_r(), 3TSOL-90
 bsltos(), 3TSOL-85
 bslundef(), 3TSOL-78
 bslvalid(), 3TSOL-89
 getcil(), 3TSOL-81
 getcsl(), 3TSOL-81
 h_alloc(), 3TSOL-90
 h_free(), 3TSOL-90
 htobcl(), 3TSOL-126
 htobclear(), 3TSOL-126
 htobil(), 3TSOL-126
 htobsl(), 3TSOL-126
 labelinfo(), 3TSOL-136
 labelvers(), 3TSOL-138
 sbcleartos(), 3TSOL-211
 sbcltos(), 3TSOL-211
 sbiltos(), 3TSOL-211
 sbsltos(), 3TSOL-211
 setbltype(), 3TSOL-88
 setcil(), 3TSOL-81

bclearhigh(), *continued*
 setcsl(), 3TSOL-81
 stobcl(), 3TSOL-217
 stobclear(), 3TSOL-217
 stobil(), 3TSOL-217
 stobsl(), 3TSOL-217
labelinfo() — information about the label encodings, 3TSOL-136
labelinfo(3TSOL), 3TSOL-7
labelvers() — **label_encodings** file version, 3TSOL-138
 library routines for client side calls
 — **clnt_call**, 3NTSOL-188
 — **clnt_freeres**, 3NTSOL-188
 — **clnt_geterr**, 3NTSOL-188
 — **clnt_perrno**, 3NTSOL-188
 — **clnt_perror**, 3NTSOL-188
 — **clnt_sperrno**, 3NTSOL-188
 — **clnt_sperror**, 3NTSOL-188
 — **rpc_broadcast**, 3NTSOL-188
 — **rpc_broadcast_exp**, 3NTSOL-188
 — **rpc_call**, 3NTSOL-188
 — **rpc_clnt_calls**, 3NTSOL-188
 library routines for dealing with creation and manipulation of CLIENT handles
 — **clnt_control**, 3NTSOL-192
 — **clnt_create**, 3NTSOL-192
 — **clnt_create_timed**, 3NTSOL-192
 — **clnt_create_vers**, 3NTSOL-192
 — **clnt_destroy**, 3NTSOL-192
 — **clnt_dg_create**, 3NTSOL-192
 — **clnt_pcreateerror**, 3NTSOL-192
 — **clnt_raw_create**, 3NTSOL-192
 — **clnt_spcreateerror**, 3NTSOL-192
 — **clnt_tli_create**, 3NTSOL-192
 — **clnt_tp_create**, 3NTSOL-192
 — **clnt_tp_create_timed**, 3NTSOL-192
 — **clnt_vc_create**, 3NTSOL-192
 — **rpc_clnt_create**, 3NTSOL-192
 — **rpc_createerr**, 3NTSOL-192
listen — listen for connections on a socket, 3NTSOL-143
 lock address space
 — **mlockall**, 3CTSOL-150

lock memory pages

- `mlock`, 3CTSOL-148

M

memory lock or unlock

- calling process — `plock`, 3CTSOL-172

memory management

- lock address space — `mlockall`, 3CTSOL-150
- lock pages in memory — `mlock`, 3CTSOL-148
- unlock address space — `munlockall`, 3CTSOL-150
- unlock pages in memory — `munlock`, 3CTSOL-148

`mldgetcwd` — get pathname of current working directory, 3TSOL-144

`mldrealpath()` — return absolute pathname, 3TSOL-146

`mldrealpathl()` — return absolute pathname, 3TSOL-146

modified functions, 3TSOL-7

N

`nftw` — walk a file tree, 3CTSOL-94

NIS+ group manipulation functions

- `nis_addmember`, 3NTSOL-152
- `nis_creategroup`, 3NTSOL-152
- `nis_destroygroup`, 3NTSOL-152
- `nis_groups`, 3NTSOL-152
- `nis_ismember`, 3NTSOL-152
- `nis_print_group_entry`, 3NTSOL-152
- `nis_removemember`, 3NTSOL-152
- `nis_verifygroup`, 3NTSOL-152

NIS+ log administration functions

- `nis_checkpoint`, 3NTSOL-161
- `nis_ping`, 3NTSOL-161

NIS+ miscellaneous functions

- `nis_freeservelist`, 3NTSOL-162
- `nis_freetags`, 3NTSOL-162
- `nis_getservlist`, 3NTSOL-162
- `nis_mkdir`, 3NTSOL-162
- `nis_rmdir`, 3NTSOL-162
- `nis_server`, 3NTSOL-162
- `nis_servstate`, 3NTSOL-162
- `nis_stats`, 3NTSOL-162

NIS+ namespace functions

- `nis_add`, 3NTSOL-155
- `nis_freeresult`, 3NTSOL-155
- `nis_lookup`, 3NTSOL-155
- `nis_modify`, 3NTSOL-155
- `nis_names`, 3NTSOL-155
- `nis_remove`, 3NTSOL-155

NIS+ table functions

- `nis_add_entry`, 3NTSOL-164
- `nis_first_entry`, 3NTSOL-164
- `nis_list`, 3NTSOL-164
- `nis_modify_entry`, 3NTSOL-164
- `nis_next_entry`, 3NTSOL-164
- `nis_remove_entry`, 3NTSOL-164
- `nis_tables`, 3NTSOL-164

`nis_tables` — NIS+ table functions, 3NTSOL-164

`nis_tables` — NIS+ table functions, 3NTSOL-164

`nis_tables` — NIS+ table functions, 3NTSOL-164

P

`plock` — lock or unlock into memory process, text, or data, 3CTSOL-172

processes

- memory lock or unlock — `plock`, 3CTSOL-172

`pututline` — access utmp file entry, 3CTSOL-119

`pututxline` — access utmpx file entry, 3CTSOL-122

R

`randomword()`

- `randomword()`, 3TSOL-175

read or write kstat data

- `kstat_read`, 3KTSOL-128
- `kstat_write`, 3KTSOL-128

remote procedure calls, library routines for — `rpc`, 3NTSOL-180

`res_init` — resolver routines, 3NTSOL-177

`res_mkquery` — resolver routines, 3NTSOL-177

`res_search` — resolver routines, 3NTSOL-177

`res_send` — resolver routines, 3NTSOL-177

`resolver` — resolver routines, 3NTSOL-177

resolver routines — `resolver`, 3NTSOL-177

- `dn_comp`, 3NTSOL-177

resolver routines — *resolver, continued*

- dn_expand, 3NTSOL-177
- res_init, 3NTSOL-177
- res_mkquery, 3NTSOL-177
- res_search, 3NTSOL-177
- res_send, 3NTSOL-177

rpc — library routines for remote procedure calls, 3NTSOL-180

RPC bind service library routines

- rpc_getallmaps, 3NTSOL-208
- rpc_getmaps, 3NTSOL-208
- rpcb_getaddr, 3NTSOL-208
- rpcb_gettime, 3NTSOL-208
- rpcb_rmtcall, 3NTSOL-208
- rpcb_set, 3NTSOL-208
- rpcb_unset, 3NTSOL-208
- rpcbind, 3NTSOL-208

RPC library routines for creation and manipulation of server handles

- rpc_svc_create, 3NTSOL-202
- svc_create, 3NTSOL-202
- svc_destroy, 3NTSOL-202
- svc_dg_create, 3NTSOL-202
- svc_fd_create, 3NTSOL-202
- svc_raw_create, 3NTSOL-202
- svc_tli_create, 3NTSOL-202
- svc_tp_create, 3NTSOL-202
- svc_vc_create, 3NTSOL-202

RPC library routines for registering servers

- rpc_reg, 3NTSOL-206
- rpc_svc_reg, 3NTSOL-206
- svc_auth_reg, 3NTSOL-206
- svc_reg, 3NTSOL-206
- svc_unreg, 3NTSOL-206
- xprt_register, 3NTSOL-206
- xprt_unregister, 3NTSOL-206

RPC servers library routines

- rpc_svc_calls, 3NTSOL-198
- svc_dg_enablecache, 3NTSOL-198
- svc_done, 3NTSOL-198
- svc_exit, 3NTSOL-198
- svc_fdset, 3NTSOL-198
- svc_freeargs, 3NTSOL-198
- svc_getargs, 3NTSOL-198
- svc_getreq_common, 3NTSOL-198

RPC servers library routines, *continued*

- svc_getreq_poll, 3NTSOL-198
- svc_getreqset, 3NTSOL-198
- svc_getrpccaller, 3NTSOL-198
- svc_pollset, 3NTSOL-198
- svc_run, 3NTSOL-198
- svc_sendreply, 3NTSOL-198

rpc_broadcast — library routines for client side calls, 3NTSOL-188

rpc_broadcast_exp — library routines for client side calls, 3NTSOL-188

rpc_call — library routines for client side calls, 3NTSOL-188

rpc_clnt_calls — library routines for client side calls, 3NTSOL-188

Routines, 3NTSOL-188

rpc_clnt_create — library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192

Routines, 3NTSOL-192

rpc_createerr — library routines for dealing with creation and manipulation of CLIENT handles, 3NTSOL-192

S

sbcleartos() — *Binary Clearance* to canonical string, 3TSOL-211

sbcלטos() — *Binary CMW Label* to canonical string, 3TSOL-211

sbiltos() — *Binary Information Label* to canonical string, 3TSOL-211

sbsltos() — *Binary Sensitivity Label* to canonical string, 3TSOL-211

security policy, 3TSOL-7

send — send message from a socket, 3NTSOL-214

sendmsg — send message from a socket, 3NTSOL-214

sendto — send message from a socket, 3NTSOL-214

set_effective_priv() — set the effective privileges for the current process., 3TSOL-216

set_inheritable_priv() — set the inheritable

privileges for the current process., 3TSOL-216
set_permitted_priv() — set the permitted
 privileges for the current process., 3TSOL-216
setac — get audit control file information,
 3TSOL-97
setauclass — rewind audit_class database file,
 3TSOL-99
setauuser — rewind audit_event database file,
 3TSOL-102
setauuser — get audit_user database entry,
 3TSOL-104
setbltype() — set type of *Binary Label*, 3TSOL-88
setcil() — replace *Information Label*, 3TSOL-81
setcsl() — replace *Sensitivity Label*, 3TSOL-81
setutent — access utmp file entry, 3CTSOL-119
setutxent — access utmpx file entry, 3CTSOL-122
socket
 accept a connection — **accept**, 3NTSOL-49
 bind a name — **bind**, 3NTSOL-72
 get options — **getsockopt**, 3NTSOL-114
 listen for connections — **listen**, 3NTSOL-143
 send message from — **send**, **sendto**,
 sendmsg, 3NTSOL-214
 set options — **setsockopt**, 3NTSOL-114
stobcl() — string to *Binary CMW Label*,
 3TSOL-217
stobclear() — string to *Binary Clearance*,
 3TSOL-217
stobil() — string to *Binary Information Label*,
 3TSOL-217
stobs1() — string to *Binary Sensitivity Label*,
 3TSOL-217
STREAMS
 accept a connection on a socket — **accept**,
 3NTSOL-49
 bind a name to a socket — **bind**, 3NTSOL-72
 get and set socket options — **getsockopt**,
 setsockopt, 3NTSOL-114
 listen for connections on a socket — **listen**,
 3NTSOL-143
 send a message from a socket — **send**,
 sendto, **sendmsg**, 3NTSOL-214

T

t_accept — accept a connect request, 3NTSOL-238
t_bind — bind an address to a transport endpoint,
 3NTSOL-240
t_optmgt — manage options for a transport end-
 point, 3NTSOL-242
t_snd — send data or expedited data over a con-
 nection, 3NTSOL-244
t_sndudata — send a data unit, 3NTSOL-246
transport functions
 accept a connect request — **t_accept**,
 3NTSOL-238
 bind an address to a transport endpoint —
 t_bind, 3NTSOL-240
 manage options for a transport endpoint —
 t_optmgt, 3NTSOL-242
 send a data unit — **t_sndudata**, 3NTSOL-246
 send data or expedited data over a connection
 — **t_snd**, 3NTSOL-244
Trusted Solaris profile entry
 — **endprofent**, 3TSOL-110
 — **free_profent**, 3TSOL-110
 — **getprofbyname**, 3TSOL-112
 — **getprofent**, 3TSOL-110
 — **getprofentbyname**, 3TSOL-110
 — **setprofent**, 3TSOL-110
Trusted Solaris profile strry
 — **endprofstr**, 3TSOL-112
 — **free_profstr**, 3TSOL-112
 — **getprofstr**, 3TSOL-112
 — **putprofstr**, 3TSOL-112
 — **setprofstr**, 3TSOL-112
Trusted Solaris user entry
 — **enduserent**, 3TSOL-117
 — **free_userent**, 3TSOL-117
 — **getuserbyname**, 3TSOL-117
 — **getuserbyuid**, 3TSOL-117
 — **getuserent**, 3TSOL-117
 — **putuserent**, 3TSOL-117
 — **setuserent**, 3TSOL-117
TSOL man page suffix, 3TSOL-7

U

unlock address space
— `munlockall`, 3CTSOL-150
unlock memory pages
— `munlock`, 3CTSOL-148
`updwtmp` — access `utmpx` file entry, 3CTSOL-122
`updwtmpx` — access `utmpx` file entry, 3CTSOL-122
user audit characteristics for socket or TLI peer
get, 3TSOL-108
`utmp` file
access entry — `getutent`, 3CTSOL-119
`utmpname` — access `utmp` file entry, 3CTSOL-119
`utmpx` file
access entry — `getutxent`, 3CTSOL-122
`utmpxname` — access `utmpx` file entry, 3CTSOL-122

V

`vfstab_adjunct` file
— `getvfsaent`, 3TSOL-125

X

`Xbclearertos()` — *Binary Clearance* to string with
font list, 3TSOL-134
`Xbsltos()` — *Binary CMW Label* to string with font
list, 3TSOL-134
`Xbiltos()` — *Binary Information Label* to string
with font list, 3TSOL-134
`Xbsltos()` — *Binary Sensitivity Label* to string with
font list, 3TSOL-134
`XTSOLgetClientAttributes` — get client attrs,
3X11TSOL-29
`XTSOLgetPropAttributes` — get all prop attrs,
3X11TSOL-30
`XTSOLgetPropLabel` — set resource label,
3X11TSOL-31
`XTSOLgetPropUID` — get property uid,
3X11TSOL-32
`XTSOLgetResAttributes` — get all attrs,
3X11TSOL-33
`XTSOLgetResLabel` — get resource label,
3X11TSOL-34
`XTSOLgetResUID` — set resource uid,

3X11TSOL-35
`XTSOLgetSSHeight` — get screen stripe height,
3X11TSOL-36
`XTSOLgetWindowIIL` — get window iil,
3X11TSOL-37
`XTSOLgetWorkstationOwner` — get ownership ,
3X11TSOL-38
`XTSOLisWindowTrusted` — test Trusted Window,
3X11TSOL-26
`XTSOLMakeTPWindow` — Make Trusted path win-
dow, 3X11TSOL-27
`XTSOLsetPolyInstInfo` — set poly instantiation
info, 3X11TSOL-39
`XTSOLsetPropLabel` — set resource label,
3X11TSOL-40
`XTSOLsetPropUID` — set property uid,
3X11TSOL-41
`XTSOLsetResLabel` — set resource label,
3X11TSOL-42
`XTSOLsetResUID` — set resource uid,
3X11TSOL-43
`XTSOLsetSessionHI` — set SL , 3X11TSOL-45
`XTSOLsetSessionLO` — set SL , 3X11TSOL-46
`XTSOLsetSSHeight` — set screen stripe height,
3X11TSOL-44
`XTSOLsetWindowIIL` — set window iil,
3X11TSOL-47
`XTSOLsetWorkstationOwner` — set ownership ,
3X11TSOL-48
`XTSOLShutdown` — shutdown system,
3X11TSOL-28