



## man pages section 3: Library Functions

---

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

Part No: 816-1057-10  
November 2001

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. All rights reserved.

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

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

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

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

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

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

---

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. Tous droits réservés

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

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

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

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

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



011029 @2471



# Contents

---

**Preface** 15

**Introduction** 21

Intro(3) 22

**Introduction to Library Functions** 37

accept(3SOCKET) 38

adornfc(3TSOL) 40

auditwrite(3TSOL) 42

au\_preselect(3BSM) 54

auth\_set\_to\_str(3TSOL) 56

auth\_to\_str(3TSOL) 57

au\_user\_mask(3BSM) 58

aw\_errno(3TSOL) 60

aw\_geterrno(3TSOL) 62

aw\_perror(3TSOL) 64

aw\_perror\_r(3TSOL) 66

aw\_strerror(3TSOL) 68

bclearhigh(3TSOL) 70

bclearlow(3TSOL) 72

bcleartoh(3TSOL) 74

bcleartoh\_r(3TSOL) 76

bcleartos(3TSOL) 78

bclearundef(3TSOL) 81

bclearvalid(3TSOL) 83

bclhigh(3TSOL) 85  
 bcllow(3TSOL) 87  
 bcltobanner(3TSOL) 89  
 bcltoh(3TSOL) 92  
 bcltoh\_r(3TSOL) 94  
 bcltos(3TSOL) 96  
 bcltosl(3TSOL) 99  
 bclundef(3TSOL) 100  
 bind(3SOCKET) 102  
 blcompare(3TSOL) 104  
 bldominates(3TSOL) 105  
 blequal(3TSOL) 106  
 blinrange(3TSOL) 107  
 blinset(3TSOL) 108  
 blmanifest(3TSOL) 110  
 blmaximum(3TSOL) 112  
 blminimum(3TSOL) 113  
 blminmax(3TSOL) 114  
 blportion(3TSOL) 115  
 blstrictdom(3TSOL) 116  
 bltcolor(3TSOL) 117  
 bltcolor\_r(3TSOL) 119  
 bltos(3TSOL) 121  
 bltype(3TSOL) 124  
 blvalid(3TSOL) 125  
 bslhigh(3TSOL) 127  
 bsllow(3TSOL) 129  
 bslttoh(3TSOL) 131  
 bslttoh\_r(3TSOL) 133  
 bsltos(3TSOL) 135  
 bslundef(3TSOL) 138  
 bsvalid(3TSOL) 140  
 btohex(3TSOL) 142  
 chkauth(3TSOL) 144  
 chkauthattr(3SECDB) 145  
 clnt\_call(3NSL) 148  
 clnt\_control(3NSL) 152

clnt_create(3NSL)	158
clnt_create_timed(3NSL)	164
clnt_create_vers(3NSL)	170
clnt_create_vers_timed(3NSL)	176
clnt_destroy(3NSL)	182
clnt_dg_create(3NSL)	188
clnt_freeres(3NSL)	194
clnt_geterr(3NSL)	198
clnt_pcreateerror(3NSL)	202
clnt_perrno(3NSL)	208
clnt_perror(3NSL)	212
clnt_raw_create(3NSL)	216
clnt_spccreateerror(3NSL)	222
clnt_sperrno(3NSL)	228
clnt_sperror(3NSL)	232
clnt_tli_create(3NSL)	236
clnt_tp_create(3NSL)	242
clnt_tp_create_timed(3NSL)	248
clnt_vc_create(3NSL)	254
clock_getres(3RT)	260
clock_gettime(3RT)	262
clock_settime(3RT)	264
dn_comp(3RESOLV)	266
dn_expand(3RESOLV)	272
door_create(3DOOR)	278
door_tcred(3DOOR)	280
endac(3BSM)	282
endauclass(3BSM)	284
endauevent(3BSM)	286
endauthattr(3SECDB)	289
endauuser(3BSM)	292
endexecattr(3SECDB)	294
endprofattr(3SECDB)	298
endprofent(3TSOL)	300
endprofstr(3TSOL)	301
enduserattr(3SECDB)	302
enduserent(3TSOL)	304

endutent(3C) 305  
 endutxent(3C) 308  
 fp\_resstat(3RESOLV) 312  
 free\_authattr(3SECDB) 318  
 free\_auth\_set(3TSOL) 321  
 free\_execattr(3SECDB) 322  
 free\_profattr(3SECDB) 326  
 free\_profent(3TSOL) 328  
 free\_profstr(3TSOL) 329  
 free\_userattr(3SECDB) 330  
 free\_userent(3TSOL) 332  
 ftw(3C) 333  
 getacdir(3BSM) 338  
 getacflg(3BSM) 340  
 getacinfo(3BSM) 342  
 getacmin(3BSM) 344  
 getacna(3BSM) 346  
 getauclassent(3BSM) 348  
 getauclassent\_r(3BSM) 350  
 getauclassnam(3BSM) 352  
 getauclassnam\_r(3BSM) 354  
 getauditflags(3BSM) 356  
 getauditflagsbin(3BSM) 357  
 getauditflagschar(3BSM) 358  
 getauevent(3BSM) 359  
 getauevent\_r(3BSM) 362  
 getauevnam(3BSM) 365  
 getauevnam\_r(3BSM) 368  
 getauevnonam(3BSM) 371  
 getauevnum(3BSM) 374  
 getauevnum\_r(3BSM) 377  
 getauthattr(3SECDB) 380  
 getauthnam(3SECDB) 383  
 get\_auth\_text(3TSOL) 386  
 getauuserent(3BSM) 387  
 getauusernam(3BSM) 389  
 getcsl(3TSOL) 391

getexecattr(3SECDB)	392
getexecprof(3SECDB)	396
getexecuser(3SECDB)	400
getfauditflags(3BSM)	404
getpeerinfo(3TSOL)	405
get_priv_text(3TSOL)	407
getprofattr(3SECDB)	409
getprofent(3TSOL)	411
getprofentbyname(3TSOL)	412
getprofnam(3SECDB)	413
getprofstr(3TSOL)	415
getprofstrbyname(3TSOL)	416
getsockopt(3SOCKET)	417
getuserattr(3SECDB)	420
getuserent(3TSOL)	422
getuserentbyname(3TSOL)	423
getuserentbyuid(3TSOL)	424
getusernam(3SECDB)	425
getuserid(3SECDB)	427
getutent(3C)	429
getutid(3C)	432
getutline(3C)	435
getutmp(3C)	438
getutmpx(3C)	442
getutxent(3C)	446
getutxid(3C)	450
getutxline(3C)	454
getvfaent(3TSOL)	458
getvfsafile(3TSOL)	460
grantpt(3C)	462
h_alloc(3TSOL)	463
herror(3RESOLV)	465
hextob(3TSOL)	471
h_free(3TSOL)	472
hstrerror(3RESOLV)	474
htobcl(3TSOL)	480
htobclear(3TSOL)	481

htobsl(3TSOL) 482  
 initgroups(3C) 483  
 kstat\_read(3KSTAT) 484  
 kstat\_write(3KSTAT) 485  
 kva\_match(3SECDB) 486  
 labelbuilder(3TSOL) 487  
 labelclipping(3TSOL) 492  
 labelinfo(3TSOL) 494  
 labelvers(3TSOL) 496  
 libt6(3NSL) 498  
 listen(3SOCKET) 502  
 match\_execattr(3SECDB) 503  
 mldgetcwd(3TSOL) 507  
 mldlstat(3TSOL) 509  
 mldrealpath(3TSOL) 510  
 mldrealpathl(3TSOL) 512  
 mldstat(3TSOL) 514  
 mlock(3C) 515  
 mlockall(3C) 517  
 munlock(3C) 519  
 munlockall(3C) 521  
 nftw(3C) 523  
 nis\_add(3NSL) 528  
 nis\_add\_entry(3NSL) 534  
 nis\_addmember(3NSL) 542  
 nis\_checkpoint(3NSL) 545  
 nis\_creategroup(3NSL) 547  
 nis\_destroygroup(3NSL) 550  
 nis\_first\_entry(3NSL) 553  
 nis\_freeresult(3NSL) 561  
 nis\_freeservlist(3NSL) 567  
 nis\_freetags(3NSL) 569  
 nis\_getservlist(3NSL) 571  
 nis\_groups(3NSL) 573  
 nis\_ismember(3NSL) 576  
 nis\_list(3NSL) 579  
 nis\_lookup(3NSL) 587



nis\_mkdir(3NSL) 593  
 nis\_modify(3NSL) 595  
 nis\_modify\_entry(3NSL) 601  
 nis\_names(3NSL) 609  
 nis\_next\_entry(3NSL) 615  
 nis\_ping(3NSL) 623  
 nis\_print\_group\_entry(3NSL) 625  
 nis\_remove(3NSL) 628  
 nis\_remove\_entry(3NSL) 634  
 nis\_removemember(3NSL) 642  
 nis\_rmdir(3NSL) 645  
 nis\_server(3NSL) 647  
 nis\_servstate(3NSL) 649  
 nis\_stats(3NSL) 651  
 nis\_tables(3NSL) 653  
 nis\_verifygroup(3NSL) 661  
 plock(3C) 664  
 priv\_set\_to\_str(3TSOL) 665  
 priv\_to\_str(3TSOL) 667  
 putprofstr(3TSOL) 669  
 pututline(3C) 670  
 pututxline(3C) 673  
 randomword(3TSOL) 677  
 res\_hostalias(3RESOLV) 679  
 res\_init(3RESOLV) 685  
 res\_mkquery(3RESOLV) 691  
 res\_nclose(3RESOLV) 697  
 res\_ninit(3RESOLV) 703  
 res\_nmkquery(3RESOLV) 709  
 res\_npquery(3RESOLV) 715  
 res\_nquery(3RESOLV) 721  
 res\_nquerydomain(3RESOLV) 727  
 res\_nsearch(3RESOLV) 733  
 res\_nsend(3RESOLV) 739  
 res\_nsendsigned(3RESOLV) 745  
 resolver(3RESOLV) 751  
 res\_query(3RESOLV) 757

res\_search(3RESOLV) 763  
 res\_send(3RESOLV) 769  
 rpc(3NSL) 775  
 rpcb\_getaddr(3NSL) 784  
 rpcb\_getallmaps(3NSL) 787  
 rpcb\_getmaps(3NSL) 790  
 rpcb\_gettime(3NSL) 793  
 rpcbind(3NSL) 796  
 rpcb\_rmtcall(3NSL) 799  
 rpc\_broadcast(3NSL) 802  
 rpc\_broadcast\_exp(3NSL) 806  
 rpcb\_set(3NSL) 810  
 rpcb\_unset(3NSL) 813  
 rpc\_call(3NSL) 816  
 rpc\_clnt\_calls(3NSL) 820  
 rpc\_clnt\_create(3NSL) 824  
 rpc\_createerr(3NSL) 830  
 rpc\_reg(3NSL) 836  
 rpc\_svc\_calls(3NSL) 839  
 rpc\_svc\_create(3NSL) 844  
 rpc\_svc\_reg(3NSL) 848  
 sbclearlos(3TSOL) 851  
 sbcltos(3TSOL) 853  
 sbltos(3TSOL) 855  
 sbsltos(3TSOL) 857  
 send(3SOCKET) 859  
 sendmsg(3SOCKET) 861  
 sendto(3SOCKET) 863  
 setac(3BSM) 865  
 setauclass(3BSM) 867  
 setauevent(3BSM) 869  
 setauthattr(3SECDB) 872  
 setauuser(3BSM) 875  
 setbltype(3TSOL) 877  
 setcsl(3TSOL) 878  
 set\_effective\_priv(3TSOL) 879  
 setexecattr(3SECDB) 881

set_inheritable_priv(3TSOL)	885
set_permitted_priv(3TSOL)	887
setprofattr(3SECDB)	889
setprofent(3TSOL)	891
setprofstr(3TSOL)	892
setsockopt(3SOCKET)	893
setuserattr(3SECDB)	896
setuserent(3TSOL)	898
setutent(3C)	899
setutxent(3C)	902
socket(3SOCKET)	906
stobcl(3TSOL)	909
stobclear(3TSOL)	912
stobl(3TSOL)	915
stobsl(3TSOL)	918
str_to_auth(3TSOL)	921
str_to_auth_set(3TSOL)	922
str_to_priv(3TSOL)	923
str_to_priv_set(3TSOL)	925
svc_auth_reg(3NSL)	927
svc_control(3NSL)	930
svc_create(3NSL)	934
svc_destroy(3NSL)	938
svc_dg_create(3NSL)	942
svc_dg_enablecache(3NSL)	946
svc_done(3NSL)	951
svc_exit(3NSL)	956
svc_fd_create(3NSL)	961
svc_fdset(3NSL)	965
svc_freeargs(3NSL)	970
svc_getargs(3NSL)	975
svc_getreq_common(3NSL)	980
svc_getreq_poll(3NSL)	985
svc_getreqset(3NSL)	990
svc_getrpccaller(3NSL)	995
svc_max_pollfd(3NSL)	1000
svc_pollfd(3NSL)	1005

svc_raw_create(3NSL)	1010
svc_reg(3NSL)	1014
svc_run(3NSL)	1017
svc_sendreply(3NSL)	1022
svc_tli_create(3NSL)	1027
svc_tp_create(3NSL)	1031
svc_unreg(3NSL)	1035
svc_vc_create(3NSL)	1038
t6alloc_blk(3NSL)	1042
t6attr_query(3NSL)	1043
t6clear_blk(3NSL)	1044
t6cmp_blk(3NSL)	1045
t6copy_blk(3NSL)	1046
t6dup_blk(3NSL)	1047
t6ext_attr(3NSL)	1048
t6free_blk(3NSL)	1049
t6get_attr(3NSL)	1050
t6get_endpt_default(3NSL)	1052
t6get_endpt_mask(3NSL)	1054
t6last_attr(3NSL)	1056
t6new_attr(3NSL)	1057
t6peek_attr(3NSL)	1058
t6recvfrom(3NSL)	1059
t6sendto(3NSL)	1061
t6set_attr(3NSL)	1063
t6set_endpt_default(3NSL)	1065
t6set_endpt_mask(3NSL)	1067
t6size_attr(3NSL)	1069
t_accept(3NSL)	1070
t_bind(3NSL)	1074
t_optmgmt(3NSL)	1078
t_snd(3NSL)	1084
t_sndudata(3NSL)	1088
tsol_lbuild_create(3TSOL)	1091
tsol_lbuild_destroy(3TSOL)	1096
tsol_lbuild_get(3TSOL)	1101
tsol_lbuild_set(3TSOL)	1106

updwtmp(3C)	1111
updwtmpx(3C)	1115
utmpname(3C)	1119
utmpxname(3C)	1122
Xbclear(3TSOL)	1126
Xbclear(3TSOL)	1128
Xbsclear(3TSOL)	1130
xprt_register(3NSL)	1132
xprt_unregister(3NSL)	1135
XTSOLgetClientAttributes(3)	1138
XTSOLgetPropAttributes(3)	1139
XTSOLgetPropLabel(3)	1140
XTSOLgetPropUID(3)	1141
XTSOLgetResAttributes(3)	1142
XTSOLgetResLabel(3)	1143
XTSOLgetResUID(3)	1144
XTSOLgetSSHheight(3)	1145
XTSOLgetWorkstationOwner(3)	1146
XTSOLisWindowTrusted(3)	1147
XTSOLmakeTPWindow(3)	1148
XTSOLsetPolyInstInfo(3)	1149
XTSOLsetPropLabel(3)	1150
XTSOLsetPropUID(3)	1151
XTSOLsetResLabel(3)	1152
XTSOLsetResUID(3)	1153
XTSOLsetSessionHI(3)	1154
XTSOLsetSessionLO(3)	1155
XTSOLsetSSHheight(3)	1156
XTSOLsetWorkstationOwner(3)	1157
XTSOLshutdown(3)	1158

<b>Index</b>	<b>1159</b>
--------------	-------------



# Preface

---

## Overview

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

## Trusted Solaris Reference Manual

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

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

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

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

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

## Man Page Sections

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

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

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

### NAME

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

### SYNOPSIS

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



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

The following special characters are used in this section:

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

#### PROTOCOL

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

#### DESCRIPTION

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

#### IOCTL

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

#### OPTIONS

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

#### OPERANDS

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

#### OUTPUT

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

#### RETURN VALUES

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

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

#### ERRORS

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

#### USAGE

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

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

#### EXAMPLES

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

#### ENVIRONMENT VARIABLES

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

#### EXIT STATUS

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

#### FILES

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

#### ATTRIBUTES

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

#### SUMMARY OF TRUSTED SOLARIS CHANGES

This section describes changes to a Solaris item by Trusted Solaris software. It is present in man pages that have been modified from Solaris software.

#### SEE ALSO

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

#### DIAGNOSTICS

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

#### WARNINGS

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

#### NOTES

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

#### BUGS

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



# Introduction

---

<b>NAME</b>	Intro – introduction to functions and libraries
<b>DESCRIPTION</b>	<p>This section describes functions found in various libraries in the Solaris and Trusted Solaris environment, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.</p> <p>These functions can be:</p> <ul style="list-style-type: none"> <li>■ Functions that are unique to and originate in the Trusted Solaris environment, such as <code>labelinfo(3tsol)</code>. <code>labelinfo()</code> gets information about security labels from the <code>label_encodings(4)</code> file.</li> <li>■ SunOS 5.8 functions and X windows functions that have been modified to work within the Trusted Solaris security policy, such as <code>accept(3SOCKET)</code>. Man pages for modified functions have been rewritten to remove information that is not accurate for how the function behaves in the Trusted Solaris environment. Modified man pages, such as <code>accept()</code>, also contain descriptions for any added features and arguments.</li> <li>■ SunOS 5.8 functions that remain unchanged from the Solaris 8 release, such as <code>connect(3SOCKET)</code>.</li> </ul> <p><b>Note</b> – The printed <i>Trusted Solaris 8 4/01 Reference Manual</i> includes only those functions that have been modified or originate in the Trusted Solaris environment. This includes X Windows Library man pages, located in <code>/usr/openwin/man/man3</code>, and the <code>dtappsession.1</code> man page, located in <code>/usr/dt/man/man1</code>. Printed versions of unchanged SunOS 5.8 man pages are found in the <i>SunOS 5.8 Reference Manual</i>.</p> <p>Function declarations can be obtained from the <code>#include</code> files indicated on each page. Pages are grouped by library and are identified by the library name (or an abbreviation of the library name) after the section number. Collections of related libraries are grouped into five volumes as described below. A sixth volume (listed first) contains pages describing the contents of each shared library and each header used by the functions, macros, and external variables described in the remaining five volumes.</p>
<b>Library Interfaces and Headers</b>	<p>This volume describes the contents of each shared library and each header used by functions, macros, and external variables described in the remaining five volumes.</p> <p>(3LIB)                      The libraries described in this section are implemented as shared objects.</p> <p>Descriptions of shared objects may include a definition of the global symbols that define the shared objects' public interface, for example <code>SUNW_1.1</code>. Other interfaces may exist within the shared object, for example <code>SUNW_private.1.1</code>. The public interface provides a stable, committed set of symbols for application development. The private interfaces are for internal use only, and may change at any time.</p>

	For many shared objects, an archive library is provided for backward compatibility on 32-bit systems only. Use of these libraries may restrict an applications ability to migrate between different Solaris releases. As dynamic linking is the preferred compilation method on Solaris, the use of these libraries is discouraged.
(3LIBUCB)	The SunOS/BSD Compatibility libraries described in this section are implemented as a shared object. See (3LIB) above.
(3HEAD)	The headers described in this section are used by functions, macros, and external variables. Headers contain function prototypes, definitions of symbolic constants, common structures, preprocessor macros, and defined types. Each function described in the remaining five volumes specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do have to be present on the target execution system.
<b>Basic Library Functions</b>	The functions described in this volume are the core C library functions that are basic to application development.
(3C)	<p>These functions, together with those of Section 2, constitute the standard C library, <code>libc</code>, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, <code>libc.so</code>, and as an archive, <code>libc.a</code>. C programs are linked with the shared object version of the standard C library by default. Specify <code>-Bstatic</code> or <code>-dn</code> on the <code>cc</code> command line to link with the archive version. See <code>libc(3LIB)</code>, <code>cc(1B)</code> for other overrides, and the “C Compilation System” chapter of the <i>ANSI C Programmer's Guide</i> for a discussion. Some functions behave differently in standard-conforming environments. This behavior is noted on the individual manual pages. See <code>standards(5)</code>.</p> <p>Some functions in <code>libc</code> have been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the individual man pages.</p>
(3DL)	These functions constitute the dynamic linking library, <code>libdl</code> . This library is implemented as a shared object, <code>libdl.so</code> , but is not automatically linked by the C compilation system. Specify <code>-ldl</code> on the <code>cc</code> command line to link with this library. See <code>libdl(3LIB)</code> .
(3MALLOC)	These functions constitute the various memory allocation libraries: <code>libmalloc</code> , <code>libbsdmalloc</code> , <code>libmapmalloc</code> , and <code>libmtmalloc</code> . Each of these libraries is implemented as a shared object ( <code>libmalloc.so</code> , <code>libbsdmalloc.so</code> , <code>libmapmalloc.so</code> , and <code>libmtmalloc.so</code> ) and all except <code>libmtmalloc</code> are

## Intro(3)

### Networking Library Functions

	<p>implemented as archives (<code>libmalloc.a</code>, <code>libbsdmalloc.a</code>, <code>libmapmalloc.a</code>). These libraries are not automatically linked by the C compilation system. Specify <code>-lmalloc</code>, <code>-libsdmalloc</code>, <code>-lmapmalloc</code>, and <code>-lmtmalloc</code> to link with, respectively, <code>libmalloc</code>, <code>libbsdmalloc</code>, <code>libmapmalloc</code>, and <code>libmtmalloc</code>. See <code>libmalloc(3LIB)</code>, <code>libbsdmalloc(3LIB)</code>, <code>libmapmalloc(3LIB)</code>, and <code>libmtmalloc(3LIB)</code>.</p>
(3UCB)	<p>These functions constitute the Source Compatibility (with BSD functions) library. It is implemented as a shared object, <code>libucb.so</code>, and as an archive, <code>libucb.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lucb</code> on the <code>cc</code> command line to link with this library, which is located in the <code>/usr/ucb</code> subdirectory. Headers for this library are located within <code>/usr/ucbinclude</code>. See <code>libucb(3LIB)</code>.</p>
	<p>The functions described in this volume comprise the various networking libraries.</p>
(3GSS)	<p>The functions in this library are the routines that comprise the Generic Security Services API library. This library is implemented as a shared object, <code>libgss.so.1</code>, but it is not automatically linked by the C compilation system. Specify <code>-lgss</code> on the <code>cc</code> command line to link with this library. See <code>libgss(3LIB)</code>.</p>
(3KRB)	<p>These functions constitute the Kerberos library <code>libkrb</code>. This library is implemented as a shared object, <code>libkrb.so</code>, and as an archive, <code>libkrb.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lkrb</code> on the <code>cc</code> command line to link with this library. See <code>libkrb(3LIB)</code>.</p>
(3LDAP)	<p>These functions constitute the Lightweight Directory Access Protocol library, <code>libldap</code>. This library is implemented as a shared object, <code>libldap.so</code>, but is not automatically linked by the C compilation system. Specify <code>-lldap</code> on the <code>cc</code> command line to link with this library. See <code>ldap(3LDAP)</code>.</p>
(3NSL)	<p>These functions constitute the Network Service Library, <code>libnsl</code>. The Trusted Solaris environment modifies some Network Service Library functions, and adds the Trusted Systems Interoperability Group (TSIG) TSIX [RE]1.1 library, <code>libt6</code>. See <code>libt6(3NSL)</code>.</p> <p><code>libnsl.so</code> and <code>libt6.so</code> are implemented as shared objects, and <code>libnsl.a</code> is also specified as an archive. Neither library is automatically linked by the C compilation system. Specify <code>-lnsl</code> on the <code>cc</code> command line to link with the <code>libnsl</code>. Specify <code>-lt6</code> on the <code>cc</code> command line to link with the <code>libt6</code> library.</p> <p>Many base networking functions are also available in the X/Open Networking Interfaces library, <code>libxnet</code>. See section (3XNET) below for more information on the <code>libxnet</code> interfaces.</p>



- (3RAC) These functions constitute the remote asynchronous calls library, `librac`. This library is implemented as a shared object, `librac.so`, and as an archive, `librac.a`, but is not automatically linked by the C compilation system. Specify `-lrac` on the `cc` command line to link with this library. See `librac(3LIB)`.
- (3RESOLV) These functions constitute the resolver library, `libresolv`. This library is implemented as a shared object, `libresolv.so`, and as an archive, `libresolv.a`, but is not automatically linked by the C compilation system. Specify `-lresolv` on the `cc` command line to link with this library. See `libresolv(3LIB)`.
- (3RPC) These functions constitute the remote procedure call libraries, `librpcsvc` and `librpcsoc`. The latter is provided for compatibility only; new applications should not link to it. Both libraries are implemented as shared objects, `librpcsvc.so` and `librpcsoc.so`, respectively, and `librpcsvc` is implemented as an archive, `librpcsvc.a`. `librt(3LIB)`. Neither library is automatically linked by the C compilation system. Specify `-lrpcsvc` or `-lrpcsoc` on the `cc` command line to link with these libraries. See `librpcsvc(3LIB)` and `librpcsoc(3LIB)`.
- (3SLP) These functions constitute the Service Location Protocol library, `libslp`. This library is implemented as a shared object, `libslp.so.1`, but it is not automatically linked by the C compilation system. See `libslp(3LIB)`.
- (3SOCKET) These functions constitute the sockets library, `libsocket`. This library is implemented as a shared object, `libsocket.so`, and as an archive, `libsocket.a`, but is not automatically linked by the C compilation system. Specify `-lsocket` on the `cc` command line to link with this library. See `libsocket(3LIB)`.
- (3XFN) These functions constitute the X/Open Federated Naming library, `libxfn`. This library is implemented as a shared object, `libxfn.so`, but is not automatically linked by the C compilation system. Specify `-lxfn` on the `cc` command line to link with this library. See `libxfn(3LIB)`, `xfn(3XFN)`, `fns(5)`, and `standards(5)`.
- (3XNET) These functions constitute X/Open networking interfaces which comply with the X/Open CAE Specification, Networking Services, Issue 4 (September, 1994). This library is implemented as a shared object, `libxnet.so`, but is not automatically linked by the C compilation system. Specify `-lxnet` on the `cc` command line to link with this library. See `libxnet(3LIB)` and `standards(5)` for compilation information.

Under all circumstances, the use of the Sockets API is recommended over the XTI and TLI APIs. If portability to other XPGV4v2 (see `standards(5)`) systems is a requirement, the application must use the `libxnet` interfaces. If portability is not

**Curses Library Functions**

required, the sockets interfaces in `libsocket` and `libnsl` are recommended over those in `libxnet`. Between the XTI and TLI APIs, the XTI interfaces (available with `libxnet`) are recommended over the TLI interfaces (available with `libnsl`).

The functions described in this volume comprise the libraries that provide graphics and character screen updating capabilities.

(3CURSES)

The functions constitute the following libraries:

`libcurses`

These functions constitute the curses library, `libcurses`. This library is implemented as a shared object, `libcurses.so`, and as an archive, `libcurses.a`, but is not automatically linked by the C compilation system. Specify `-lcurses` on the `cc` command line to link with this library. See `libcurses(3LIB)`.

`libform`

These functions constitute the forms library, `libform`. This library is implemented as a shared object, `libform.so`, and as an archive, `libforms.a`, but is not automatically linked by the C compilation system. Specify `-lform` on the `cc` command line to link with this library. See `libform(3LIB)`.

`libmenu`

These functions constitute the menus library, `libmenu`. This library is implemented as a shared object, `libmenu.so`, and as an archive, `libmenu.a`, but is not automatically linked by the C compilation system. Specify `-lmenu` on the `cc` command line to link with this library. See `libmenu(3LIB)`.

`libpanel`

These functions constitute the panels library, `libpanel`. This library is implemented as a shared object, `libpanel.so`, and as an archive, `libpanel.a`, but is not automatically linked by the C compilation system. Specify `-lpanel` on the `cc` command line to link with this library. See `libpanel(3LIB)`.

(3PLOT)

These functions constitute the graphics library, `libplot`. This library is implemented as a shared object, `libplot.so`, and as an archive, `libplot.a`, but is not automatically linked by the C compilation system. Specify `-lplot` on the `cc` command line to link with this library. See `libplot(3LIB)`.

(3XCURSES)

These functions constitute the X/Open Curses library, located in `/usr/xpg4/lib/libcurses.so.1`. This library provides a set

## Threads and Realtime Library Functions

	of internationalized functions and macros for creating and modifying input and output to a terminal screen. Included in this library are functions for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor. X/Open Curses is designed to optimize screen update activities. The X/Open Curses library conforms fully with Issue 4 of the X/Open Extended Curses specification.
The functions described in this volume constitute the threads and realtime libraries.	
(3AIO)	These functions constitute the asynchronous I/O library, <code>liaio</code> . This library is implemented as a shared object, <code>libaio.so</code> , but is not automatically linked by the C compilation system. Specify <code>-laio</code> on the <code>cc</code> command line to link with this library. See <code>libaio(3LIB)</code> .
(3DOOR)	These functions constitute the doors library, <code>libdoor</code> . This library is implemented as a shared object, <code>libdoor.so</code> , but is not automatically linked by the C compilation system. Specify <code>-ldoor</code> on the <code>cc</code> command line to link with this library.  The Trusted Solaris environment adds the function <code>door_tcred()</code> to <code>ldoor</code> and modifies the <code>door_create()</code> function. Changes in behavior or requirements are noted on the individual man pages.
(3RT)	These functions constitute the POSIX.4 Realtime library, <code>librt</code> . It is implemented as a shared object, <code>librt.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lrt</code> on the <code>cc</code> command line to link with this library. Note that the former name for this library, <code>libposix4</code> , is maintained for backward compatibility but should be avoided. See <code>librt(3LIB)</code>  The <code>clock_settime()</code> function in <code>librt</code> has been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the man page.
(3SCHED)	These functions constitute the LWP scheduling library, <code>libsched</code> . This library is implemented as a shared object, <code>libsched.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lsched</code> on the <code>cc</code> command line to link with this library. .
(3THR)	These functions constitute the threads libraries, <code>libpthread</code> , <code>libthread</code> , and <code>libthread_db</code> . The <code>libpthread</code> and <code>libthread</code> libraries are used for building multithreaded applications: <code>libpthread</code> implements the POSIX (see <code>standards(5)</code> ) threads interface, whereas <code>libthread</code> implements the Solaris threads interface. The <code>libthread_db</code> library is useful for building debuggers for multithreaded applications.

Both POSIX threads and Solaris threads can be used within the same application. Their implementations are completely compatible with each other; however, only POSIX threads guarantee portability to other POSIX-conforming environments.

When POSIX and Solaris threads are used in the same application, if there are calls with the same name but different semantics, the POSIX semantic supersedes the Solaris threads semantic. For example, the call to `fork()` will imply the `fork1()` semantic in a program linked with the POSIX threads library, whether or not it is also linked with `-lthread` (Solaris threads).

The `libpthread`, `libthread`, and `libthread_db` libraries are implemented as shared objects, `libpthread.so`, `libthread_db.so`, and `libthread.so`, respectively, but only `libthread_db` is implemented as an archive library, `libthread_db.a`. These libraries are not automatically linked by the C compilation system. Specify `-lpthread`, `-lthread`, or `-lthread_db` on the `cc` command line to link with these libraries. See `libpthread(3LIB)`, `libthread(3LIB)`, and `libthread_db(3LIB)`.

The following functions are optional under POSIX and are not supported in the current Solaris release.

```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
                                int protocol);

int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *attr,
                                int *protocol);

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
                                    int prioceiling);

int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr,
                                    int *prioceiling);
```

## Extended Library Functions

The functions described in this volume comprise various specialized libraries that are not limited to the following:

### (3BSM)

These functions constitute the basic security library, `libbsm`. This library is implemented as a shared object, `libbsm.so`, and as an archive, `libbsm.a`, but is not automatically linked by the C compilation system. Specify `-lbsm` on the `cc` command line to link with this library. See `libbsm(3LIB)`.

The Trusted Solaris environment modifies some functions in `libbsm`. Changes in behavior or requirements are noted on the individual man pages.

(3CFGADM)	These functions constitute the configuration administration library, <code>libcfgadm</code> . This library is implemented as a shared object, <code>libcfgadm.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lcfgadm</code> on the <code>cc</code> command line to link with this library. See <code>libcfgadm(3LIB)</code> .
(3CPC)	These functions constitute the CPU performance counter library, <code>libcpc</code> , and the process context library, <code>libpctx</code> . These libraries are implemented as shared objects, <code>libcpc.so</code> and <code>libpctx.so</code> , respectively, but are not automatically linked by the C compilation system. Specify <code>-lcpc</code> or <code>-lpctx</code> on the <code>cc</code> command line to link with these libraries. See <code>libcpc(3LIB)</code> and <code>libpctx(3LIB)</code> .
(3DEVID)	These functions constitute the device ID library, <code>libdevid</code> . This library is implemented as a shared object, <code>libdevid.so</code> , but is not automatically linked by the C compilation system. Specify <code>-ldevid</code> on the <code>cc</code> command line to link with this library. See <code>libdevid(3LIB)</code> .
(3DEVINFO)	These functions constitute the device information library, <code>libbsm</code> . This library is implemented as a shared object, <code>libdevinfo.so</code> , and as an archive, <code>libdevinfo.a</code> , but is not automatically linked by the C compilation system. Specify <code>-ldevinfo</code> on the <code>cc</code> command line to link with this library. See <code>libdevinfo(3LIB)</code> .
(3DMI)	These functions constitute the DMI libraries, <code>libdmi</code> , <code>libdmici</code> , and <code>libdmimi</code> . These libraries are implemented as shared objects, <code>libdmi.so</code> , <code>libdmici.so</code> , and <code>libdmimi.so</code> , respectively, but are not automatically linked by the C compilation system. Specify <code>-ldmi</code> , <code>-ldmici</code> , or <code>-ldmimi</code> on the <code>cc</code> command line to link with these libraries. See <code>libdmi(3LIB)</code> , <code>libdmici(3LIB)</code> , and <code>libdmimi(3LIB)</code> .
(3ELF)	These functions constitute the ELF access library, <code>libelf</code> , (Extensible Linking Format). This library provides the interface for the creation and analyses of “elf” files; executables, objects, and shared objects. <code>libelf</code> is implemented as a shared object, <code>libelf.so</code> , and as an archive, <code>libelf.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lelf</code> on the <code>cc</code> command line to link with this library. See <code>libelf(3LIB)</code> .
(3EXACCT)	These functions constitute the extended accounting access library, <code>libexacct</code> , and the project database access library, <code>libproject</code> . These libraries are implemented as shared objects, <code>libexacct.so</code> and <code>libproject.so</code> , respectively, but are not automatically linked by the C compilation system. Specify <code>-lexacct</code> or <code>-lproject</code> on the <code>cc</code> command line to link with these libraries. See <code>libexacct(3LIB)</code> and <code>libproject(3LIB)</code> .

## Intro(3)

(3GEN)	These functions constitute the string pattern-matching and pathname manipulation library, <code>libgen</code> . This library is implemented as a shared object, <code>libgen.so</code> , and as an archive, <code>libgen.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lgen</code> on the <code>cc</code> command line to link with this library. See <code>libgen(3LIB)</code> .
(3KSTAT)	<p>These functions constitute the kernel statistics library, which is implemented as a shared object, <code>libkstat.so</code>, and as an archive, <code>libkstat.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lkstat</code> on the <code>cc</code> command line to link with this library. See <code>libkstat(3LIB)</code>.</p> <p>The <code>kstat_read()</code> function in <code>libkstat</code> has been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the man page.</p>
(3KVM)	<p>These functions allow access to the kernel's virtual memory library, which is implemented as a shared object, <code>libkvm.so</code>, and as an archive, <code>libkvm.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lkvm</code> on the <code>cc</code> command line to link with this library. See <code>libkvm(3LIB)</code>.</p> <p>The <code>kstat_write()</code> function in <code>libkvm</code> has been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the man page.</p>
(3LAYOUT)	These functions constitute the layout service library, which is implemented as a shared object, <code>liblayout.so</code> , but is not automatically linked by the C compilation system. Specify <code>-llayout</code> on the <code>cc</code> command line to link with this library. See <code>liblayout(3LIB)</code> .
(3M)	These functions constitute the mathematical library, <code>libm</code> . This library is implemented as a shared object, <code>libm.so</code> , and as an archive, <code>libm.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lm</code> on the <code>cc</code> command line to link with this library.
(3MAIL)	These functions constitute the user mailbox management library, <code>libmail</code> . This library is implemented as a shared object, <code>libmail.so</code> , and as an archive, <code>libmail.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lmail</code> on the <code>cc</code> command line to link with this library.
(3MP)	These functions constitute the integer mathematical library, <code>libmp</code> . This library is implemented as a shared object, <code>libmp.so</code> , and as an archive, <code>libmp.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lmp</code> on the <code>cc</code> command line to link with this library. See <code>libmp(3LIB)</code> .

(3NVPAIR)	These functions constitute the name–value pair library, <code>libnvpair</code> . This library is implemented as a shared object, <code>libnvpair.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lnvpair</code> on the <code>cc</code> command line to link with this library. See <code>libnvpair(3LIB)</code> .
(3PAM)	These functions constitute the Pluggable Authentication Module (PAM) library, <code>libpam</code> . This library is implemented as a shared object, <code>libpam.so</code> , and as an archive, <code>libpam.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lpam</code> on the <code>cc</code> command line to link with this library. See <code>libpam(3LIB)</code> .
(3PICL)	These functions constitute the PICL library, <code>libpicl</code> . This library is implemented as a shared object, <code>libpicl.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lpicl</code> on the <code>cc</code> command line to link with this library. See <code>libpicl(3LIB)</code> and <code>libpicl(3PICL)</code> .
(3PICLTREE)	These functions constitute the PICL plug-in library, <code>libpicltree</code> . This library is implemented as a shared object, <code>libpicltree.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lpicltree</code> on the <code>cc</code> command line to link with this library. See <code>libpicltree(3LIB)</code> and <code>libpicltree(3PICLTREE)</code> .
(3SEC)	These functions constitute the file access control library, <code>libsec</code> . This library is implemented as a shared object, <code>libsec.so</code> , and as an archive, <code>libsec.a</code> , but is not automatically linked by the C compilation system. Specify <code>-lsec</code> on the <code>cc</code> command line to link with this library. See <code>libsec(3LIB)</code> .
(3SECDB)	These functions constitute the security attributes database library, <code>libsecdb</code> . This library is implemented as a shared object, <code>libsecdb.so</code> , but is not automatically linked by the C compilation system. Specify <code>-lsecdb</code> on the <code>cc</code> command line to link with this library. See <code>libsecdb(3LIB)</code> .
	The Trusted Solaris environment adds some functions to <code>libsecdb</code> and modifies others. Changes in behavior or requirements are noted on the individual man pages.
(3SNMP)	These functions constitute the SNMP libraries, <code>libdssagent</code> and <code>libdssasnmplib</code> . These libraries are implemented as shared objects, <code>libssagent.so</code> and <code>libssasnmplib.so</code> , respectively, but are not automatically linked by the C compilation system. Specify <code>-lssagent</code> or <code>-lssasnmplib</code> on the <code>cc</code> command line to link with these libraries. See <code>libssagent(3LIB)</code> and <code>libssasnmplib(3LIB)</code> .
(3SYSEVENT)	These functions constitute the system event library, <code>libsysevent</code> . This library is implemented as a shared object, <code>libsysevent.so</code> ,

## Intro(3)

	<p>but is not automatically linked by the C compilation system. Specify <code>-lsysevent</code> on the <code>cc</code> command line to link with this library. See <code>libsysevent(3LIB)</code>.</p>
(3TNF)	<p>These functions constitute the TNF libraries, <code>libtnf</code>, <code>libtnfctl</code>, and <code>libtnfprobe</code>. These libraries are implemented as shared objects, <code>libtnf.so</code>, <code>libtnfctl.so</code>, and <code>libtnfprobe.so</code>, respectively, but are not automatically linked by the C compilation system. Specify <code>-ltnf</code>, <code>-ltnfctl</code>, or <code>-ltnfprobe</code> on the <code>cc</code> command line to link with these libraries. See <code>libtnfctl(3TNF)</code>, <code>libtnfctl(3LIB)</code>, and <code>libtnfprobe(3LIB)</code>.</p>
(3VOLMGT)	<p>These functions constitute the volume management library, <code>libvolmgt</code>. This library is implemented as a shared object, <code>libvolmgt.so</code>, and as an archive, <code>libvolmgt.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lvolmgt</code> on the <code>cc</code> command line to link with this library. See <code>libvolmgt(3LIB)</code>.</p>
(3WSREG)	<p>These functions constitute the product install registry library, <code>libwsreg</code>. This library is implemented as a shared object, <code>libwsreg.so</code>, but is not automatically linked by the C compilation system. Specify <code>-lwsreg</code> on the <code>cc</code> command line to link with this library. See <code>libwsreg(3LIB)</code>.</p>
(3TSOL)	<p>These functions constitute the Trusted Solaris library <code>libtsol</code>. <code>libtsol.so</code> is implemented as a shared object but is not automatically linked by the C compilation system. To link with the <code>libtsol</code> library specify <code>-ltsol</code> on the <code>cc</code> command line.</p>
(3X11)	<p>The printed and AnswerBook2 versions of the Trusted Solaris 8 4/01 Reference Manual include these functions, which constitute the Trusted Solaris extension to the X windows library <code>libXtsol</code>. <code>libXtsol.so</code> is implemented as a shared object but is not automatically linked by the C compilation system. To link with the <code>libXtsol</code> library, specify <code>-lX11</code> and then <code>-lXtsol</code> on the <code>cc</code> command line (<code>cc -lX11 -lXtsol</code>).</p> <p>Online man pages for these functions are stored in <code>/usr/openwin/man/man3</code>.</p>
<b>SECURITY POLICY</b>	<p>System calls enforce policy for library routines, and you should generally look to the system call man page for the 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:</p> <ul style="list-style-type: none"> <li>■ If the relationship between the library routine and the underlying system call is intuitively obvious, as is the relationship between <code>fopen(3UCB)</code> and <code>open(2)</code>, the related system call is mentioned in the <b>SEE ALSO</b> section, and the policy is not repeated on the library routine's man page.</li> </ul>



- If the relationship between the library routine and the underlying system call(s) is not obvious, the policy information appears on the library routine's man page.
- If 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(3NSL)`, which relies on `streamio(7I)`, whose man page is 21 pages.
- If the library is the exposed interface, and if the system call is undocumented, the policy appears on the library man page. One example of this is in the TSIX library routines, some of which rely on undocumented system calls.

**DEFINITIONS**

A character is any bit pattern able to fit into a byte on the machine. In some international languages, however, 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.

**MT-Level of Libraries FILES**

See `attributes(5)` for descriptions of library MT-Levels.

`INCDIR` usually `/usr/include`

`LIBDIR` usually `/usr/lib` (32-bit) or  
`/usr/lib/sparcv9` (64-bit)

`LIBDIR/libc.so`

`LIBDIR/libc.a`

`LIBDIR/libgen.a`

`LIBDIR/libm.a`

`LIBDIR/libsfm.sa`

`/usr/lib/libc.so.1`

**Trusted Solaris 8 4/01 Reference Manual**

`ld(1)`, `fork(2)`

For assistance specific to Trusted Solaris libraries, see `intro(2)`, specifically the **DEFINITIONS** section, and the *Trusted Solaris Developer's Guide*.

**SunOS 5.8 Reference Manual**

`ar(1)`, `cc(1B)`, `stdio(3C)`, `attributes(5)`, `standards(5)`

*Linker and Libraries Guide*

*Profiling Tools*

*ANSI C Programmer'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 MULTITHREADED 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-conforming 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 -lthread
```

For Solaris threads behavior, compile as follows:

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

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.

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

MT-Safe interfaces are denoted in the `ATTRIBUTES` section of the functions and libraries manual pages (see `attributes(5)`). If a manual page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.

## REALTIME APPLICATIONS

Be sure to have set the environment variable `LD_BIND_NOW` to a non-null value to enable early binding. Refer to the "When Relocations are Processed" chapter in *Linker and Libraries Guide* for additional information.

## NOTES

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 `-l` option to `lint`. (For example, `-lm` includes definitions for `libm`.) Use of `lint` is highly recommended. See the `lint` chapter in *Performance 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`.

Intro(3)

# Introduction to Library Functions

---

accept(3SOCKET)

NAME	accept – accept a connection on a socket								
SYNOPSIS	<pre><b>cc</b> [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int <b>accept</b>(int s, struct sockaddr *addr, socklen_t *addrlen);</pre>								
DESCRIPTION	<p>The argument <i>s</i> is a socket that has been created with <code>socket(3SOCKET)</code> and bound to an address with <code>bind(3SOCKET)</code>, and that is listening for connections after a call to <code>listen(3SOCKET)</code>. The <code>accept()</code> function 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 non-blocking, <code>accept()</code> blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, <code>accept()</code> returns an error as described below. The <code>accept()</code> function uses the <code>netconfig(4)</code> 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>; it is not used to accept more connections. The original socket (<i>s</i>) remains open for accepting further connections.</p> <p>The argument <i>addr</i> is a result parameter that is filled in 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.</p> <p>The argument <i>addrlen</i> is a value-result parameter. Initially, it contains the amount of space pointed to by <i>addr</i>; on return it contains the length in bytes of the address returned.</p> <p>The <code>accept()</code> function is used with connection-based socket types, currently with <code>SOCK_STREAM</code>.</p> <p>It is possible to <code>select(3C)</code> or <code>poll(2)</code> a socket for the purpose of an <code>accept()</code> by selecting or polling it for a read. However, this will only indicate when a connect indication is pending; it is still necessary to call <code>accept()</code>.</p>								
RETURN VALUES	The <code>accept()</code> function returns <code>-1</code> on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.								
ERRORS	<p><code>accept()</code> will fail if:</p> <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>EMFILE</td><td>The per-process descriptor table is full.</td></tr><tr><td>ENODEV</td><td>The protocol family and type corresponding to <i>s</i> could not be found in the <code>netconfig</code> file.</td></tr></table>	EBADF	The descriptor is invalid.	EINTR	The accept attempt was interrupted by the delivery of a signal.	EMFILE	The per-process descriptor table is full.	ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the <code>netconfig</code> file.
EBADF	The descriptor is invalid.								
EINTR	The accept attempt was interrupted by the delivery of a signal.								
EMFILE	The per-process descriptor table is full.								
ENODEV	The protocol family and type corresponding to <i>s</i> could not be found in the <code>netconfig</code> file.								

accept(3SOCKET)

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 non-blocking and no connections are present to be accepted.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**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(3SOCKET) for more information.

**Trusted Solaris 8 4/01 Reference Manual** bind(3SOCKET), listen(3SOCKET), socket(3SOCKET)  
**Trusted Solaris 8 4/01 Reference Manual** poll(2), select(3C), socket(3HEAD), connect(3SOCKET), netconfig(4), attributes(5)

## adornfc(3TSOL)

<b>NAME</b>	adornfc – Adorn the final component of a pathname																
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -ltsol [library...]  #include &lt;tsol/mld.h&gt;  int <b>adornfc</b>(char *path_name, char *adorned_name);</pre>																
<b>DESCRIPTION</b>	adornfc() adorns the final component of <i>path_name</i> unless it is already adorned. <i>path_name</i> is a pathname to a filesystem 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 MAXPATHLEN bytes in length.																
<b>RETURN VALUES</b>	<p>adornfc() returns:</p> <p>0            On success.</p> <p>-1           On failure and sets errno to indicate the error.</p>																
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe										
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
Availability	SUNWtsu																
MT-Level	MT-Safe																
<b>ERRORS</b>	<p>adornfc() fails if one or more of the following are true:</p> <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 MAXPATHLEN.</td></tr> <tr> <td></td><td>A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).</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 MAXPATHLEN.		A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).	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 MAXPATHLEN.																
	A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).																
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.																



pathconf(2)  
attributes(5)

adornfc(3TSOL)

## auditwrite(3TSOL)

<b>NAME</b>	auditwrite – construct and write user-level audit records
<b>SYNOPSIS</b>	<pre><b>cc</b> [<i>flag...</i>] <i>file...</i> -l<code>bsm</code> -l<code>socket</code> -l<code>ns</code> -l<code>intl</code> -l<code>tsol</code> [<i>library...</i>]  #include &lt;bsm/auditwrite.h&gt; #include &lt;bsm/audit_uevents.h&gt; #include &lt;tsol/priv.h&gt; #include &lt;tsol/label.h&gt;  int <b>auditwrite</b>(..., AW_END);</pre>
<b>DESCRIPTION</b>	<p><code>auditwrite()</code> provides a single-function programmer interface to auditing for user-level programs. The principal features of <code>auditwrite()</code> 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 and for multithreading considerations.</p>
<b>Record Construction</b>	<p><code>auditwrite()</code> 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 <code>auditwrite()</code>. Multi-shot audit records are those constructed piecemeal through two or more calls to <code>auditwrite()</code>. See EXAMPLES.</p>
<b>Record Queueing</b>	<p>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.</p>
<b>Save Area</b>	<p>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 <code>auditwrite()</code>.</p>
<b>Trusted Server Auditing Support</b>	<p>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 <code>AW_SERVER</code> command tells <code>auditwrite()</code> 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.</p>
<b>PARAMETERS</b>	<p><code>auditwrite()</code> takes a variable number of arguments. Arguments are of three types. One type, referred to as control commands, controls the behavior of <code>auditwrite()</code>. Only one normal control command may appear in a single <code>auditwrite()</code> invocation.</p> <p>(By default, control commands apply to all record descriptors. <code>AW_USERD</code> is a special control command which can affect command scope and can be used with other control commands; see the <code>AW_USERD</code> command for details.)</p> <p>Another type of argument is an attribute command. Attribute commands describe the attributes that comprise an audit record. The last type of argument is the terminator command. The terminator command notifies <code>auditwrite()</code> when to stop parsing the invocation line.</p>

<b>Control Commands</b>	AW_WRITE	Write to the audit trail the default audit record or the audit record specified by AW_USERD. 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 <i>rd</i>	Discard the record descriptor specified by <i>rd</i> . An <i>rd</i> of -1 can be specified to discard the default <i>rd</i> .
	AW_GETRD, int * <i>rd</i>	Obtain an audit record descriptor for use with AW_USERD.
	AW_PRESELECT, au_mask_t * <i>pmask</i>	Preselect audit records according to <i>pmask</i> . Normally audit records are preselected by auditwrite() using the preselection mask in the execution environment. See getaudit(2) 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 <i>hi_water</i>	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 <i>hi_water</i> , 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 * <i>rd</i>	Turn on use of a save area. Attributes stored in the save area are prepended to records before they are written.

## auditwrite(3TSOL)

	<p>A new record descriptor is obtained and returned in <i>rd</i>, in the same way as <code>AW_GETRD</code>, and this is marked as the active save area.</p> <p>This descriptor can be used with <code>AW_USERD</code> and <code>AW_APPEND</code> to accumulate attributes in the save area.</p> <p><code>AW_SAVERD</code> should not be used on the same call with <code>AW_USERD</code>.</p>
<code>AW_NOSAVE</code>	Turn off use of the save area.
<code>AW_SERVER</code>	Turn on the trusted server option. The <code>AW_SERVER</code> command tells <code>auditwrite()</code> that the calling program is a server that must generate complete audit records. <code>auditwrite()</code> adds header and trailer attributes to all records. Sequence and group attributes are also added depending upon the audit policy. See <code>audit(2)</code> for further information on audit policy.
<code>AW_NOSERVER</code>	Turn off the trusted server option.
<code>AW_USERD, int rd</code>	<p>Use the record descriptor obtained with <code>AW_GETRD</code>. This command can be used with other control commands except <code>AW_DEFAULTRD</code>, <code>AW_DISCARD</code>, <code>AW_DISCARDRD</code>, <code>AW_FLUSH</code>, and <code>AW_GETRD</code>.</p> <p>The record descriptor and any related context is automatically released upon completion of the <code>AW_WRITE</code> command or upon any error.</p> <p>When used on the same call with other commands, this <i>must</i> be the first argument. This command will limit the scope of the other commands to the specified record descriptor. Subsequent calls are not affected unless they also use <code>AW_USERD</code> with the same descriptor, along with another command.</p> <p>In this mode, the descriptor is not affected by other <code>auditwrite</code> calls. For instance:</p> <pre>auditwrite(AW_PRESELECT, &amp;mymask, AW_END);  auditwrite(AW_USERD, rdn, AW_WRITE, ..., AW_END);</pre> <p>The first call will not change the behavior of the second, because the descriptor <i>rdn</i> has its own separate context.</p>

**Attribute  
Commands**

When used as the only command, this changes the scope of `AW_WRITE` and `AW_APPEND` commands on all subsequent calls which don't include `AW_USERD`.

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, uint32_t 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(2)` 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_INT32
AWD_INT64
AWD_INT (This is provided for compatibility but should otherwise not be used.
It is the equivalent of AWD_INT32.)
AWD_LONG (This is provided for compatibility but should otherwise not be used.
It is the equivalent of AWD_INT32.)
```

*unit\_count* describes how many elements of *unit\_type* exist in *p*, which is an address pointing to the data to be written.

## auditwrite(3TSOL)

**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(4)`. (**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(4)` and `</usr/include/bsm/audit_uevent.h>`. See `audit_event(4)` 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(2)` 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 (*arge*) from `auditon(2)` 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(1M)`) is not configured for including supplementary groups, the command is ignored.

**AW\_INADDR**, struct in\_addr \**in\_addr*

Place the specified Internet address into the audit record. (This is provided for compatibility but should otherwise not be used. It is the equivalent of **AW\_IN\_ADDR**.)

**AW\_IN\_ADDR**, struct in\_addr \**in\_addr*

Place the specified Internet address into the audit record.

**AW\_IN\_ADDR\_EX**, struct in6\_addr \**in6\_addr*

Place the specified multi-format 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 *ipport*

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(2)* 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, and the *AW\_PROCESS\_EX* and *AW\_SUBJECT\_EX* attributes, record the same information. Use the *AW\_PROCESS* or *AW\_PROCESS\_EX* attribute when recording information about a process object. Use the *AW\_SUBJECT* or *AW\_SUBJECT\_EX* attribute when recording information about a process subject. *AW\_PROCESS\_EX* and *AW\_SUBJECT\_EX* are extended forms which support multiple IP address formats in the terminal ID field.

*AW\_PROCESS\_EX*, 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. See *AW\_PROCESS*.

## auditwrite(3TSOL)

**AW\_RETURN**, char *number*, u\_int *retval*

Indicates the success or failure of an audit event. This attribute is used by `auditwrite()` for preselection, and by the `auditreduce(1M)` 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(1M)` 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(2)` 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\_asid\_t *sid*, au\_tid\_t *\*tid*

Place the specified subject information into the audit record. See `AW_PROCESS`.

**AW\_SUBJECT\_EX**, 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\_addr\_t *\*tid*

Place the specified subject information into the audit record. See `AW_PROCESS`.

**AW\_TEXT**, char *\*text*

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

**AW\_UAUTH**, char *\*auth*

Place the specified authorization name into the audit record.

**AW\_USEOFPRIV**, 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**, uint32\_t *xid*, uid\_t *creator\_uid*

Place the specified X colormap information into the audit record.

**AW\_XCLIENT**, uint32\_t *clientid*

Place the specified X client ID into the audit record.

**AW\_XCURSOR**, uint32\_t *xid*, uid\_t *creator\_uid*

Place the specified X cursor information into the audit record.

**AW\_XFONT**, uint32\_t *xid*, uid\_t *creator\_uid*

Place the specified X font information into the audit record.



	<p><code>AW_XGC, uint32_t xid, uid_t creator_uid</code> Place the specified X gc information into the audit record.</p> <p><code>AW_XPIXMAP, uint32_t xid, uid_t creator_uid</code> Place the specified X pixel-mapping information into the audit record.</p> <p><code>AW_XPROPERTY, uint32_t xid, uid_t creator_uid, char *atom_name</code> Place the specified X property information into the audit record.</p> <p><code>AW_XSELECT, char *property_string, char *property_type, char *window_data</code> Place the specified X select information into the audit record.</p> <p><code>AW_XWINDOW, uint32_t xid, uid_t creator_uid</code> Place the specified X window information into the audit record.</p>						
<b>Terminator Command</b>	The terminator command <code>AW_END</code> must be the last argument on the <code>auditwrite()</code> invocation line.						
<b>ATTRIBUTES</b>	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWcsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe						
<b>RETURN VALUES</b>	<p><code>auditwrite()</code> returns:</p> <p>0            On success.</p> <p>-1           On failure, and sets <code>aw_errno</code> to indicate the error.</p>						
<b>ERRORS</b>	<p>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 <code>LOG_ALERT</code> message is sent to <code>syslogd(1M)</code> and an attempt is made to write an <code>auditwrite()</code> “processing error” audit record to the audit trail. All record descriptors and related context are freed also.</p> <p><code>aw_strerror(3TSOL)</code> or <code>aw_perror(3TSOL)</code> may be used for obtaining the error strings associated with <code>aw_errno</code>.</p> <p>When multiple threads may be auditing, <code>aw_errno</code> may not be retrieved in an MT-safe way. In this environment, <code>AW_USERD</code> should be used on all <code>auditwrite</code> calls (after a <code>AW_GETRD</code> call), and <code>aw_geterrno(3TSOL)</code>, <code>aw_strerror(3TSOL)</code>, and <code>aw_perror_r(3TSOL)</code> should be used to obtain and format error information.</p> <p>These are the possible values of <code>aw_errno</code>:</p> <table><tr><td><code>AW_ERR_ADDR_INVALID</code></td><td>An address specified was invalid.</td></tr><tr><td><code>AW_ERR_ALLOC_FAIL</code></td><td>An attempt to allocate memory failed.</td></tr></table>	<code>AW_ERR_ADDR_INVALID</code>	An address specified was invalid.	<code>AW_ERR_ALLOC_FAIL</code>	An attempt to allocate memory failed.		
<code>AW_ERR_ADDR_INVALID</code>	An address specified was invalid.						
<code>AW_ERR_ALLOC_FAIL</code>	An attempt to allocate memory failed.						

## auditwrite(3TSOL)

AW_ERR_AUDITON_FAIL	The auditon(2) system call failed. See <code>errno</code> for the reason.
AW_ERR_AUDIT_FAIL	The audit(2) system call failed. See <code>errno</code> for the reason.
AW_ERR_CMD_INCOMPLETE	A required command was omitted. This event occurs when <code>AW_APPEND</code> 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 <code>AW_QUEUE</code> or <code>AW_SAVERD</code> 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 <code>getaudit(2)</code> or <code>getaudit_addr(2)</code> system call failed. See <code>errno</code> 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.

### Valid Examples

#### EXAMPLE 1 Single-shot record construction

```
/* Single-shot record construction:
 * Construct an audit record and write the record to the audit trail.
 * Uses the default audit record.
 */
```

**EXAMPLE 1** Single-shot record construction     *(Continued)*

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

**EXAMPLE 2** Multi-shot construction

```
/* 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);
```

**EXAMPLE 3** Multi-shot record construction

```
/* 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);
```

**EXAMPLE 4** Queueing

```
/* Queueing:
 * 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);
```

auditwrite(3TSOL)

**EXAMPLE 4** Queueing      *(Continued)*

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

**EXAMPLE 5** Using record descriptors

```
/*
 * Note that this is not MT-safe; see next example for
 * correct multithreaded code.
 *
 * Note that after the WRITE, the record descriptor is freed.
 * For subsequent auditing the AW_GET needs to be repeated
 * before record appending/writing.
 */
(void) auditwrite(AW_GETRD, &rdn, AW_END);
(void) auditwrite(AW_USERD, rdn, AW_END);

(void) auditwrite(AW_APPEND, AW_TEXT, "part 1", AW_END);
(void) auditwrite(AW_EVENTNUM, event_no,
    AW_WRITE, AW_TEXT, "part 2", AW_END);
```

**EXAMPLE 6** Multithreading

```
/*
 * This approach will be safe even when multiple threads may
 * be auditing...
 *
 * Note that after the WRITE, the record descriptor and all
 * context is freed. For subsequent auditing the entire
 * sequence needs to be repeated: get an rd, set state (such
 * as PRESELECT, SERVER, ...), and record appending/writing.
 */
(void) auditwrite(AW_GETRD, &rdn, AW_END);
(void) auditwrite(AW_USERD, rdn, AW_PRESELECT, &mymask, AW_END);
(void) auditwrite(AW_USERD, rdn, AW_APPEND, AW_TEXT, "part 1", AW_END);
(void) auditwrite(AW_USERD, rdn, AW_EVENTNUM, event_no,
    AW_WRITE, AW_TEXT, "part 2", AW_END);
```

**Invalid Examples**

**EXAMPLE 7** Specify no more than one normal control command

```
/*
 * Invalid command combinations:
 * Only one normal control command may be specified.
 */
(void) auditwrite(AW_EVENTNUM, valid_event_str, AW_TEXT, "text",
    AW_DISCARD, AW_WRITE, AW_END);
```

**EXAMPLE 8** No control command specified

```
/*
 * Invalid command combinations:
 * No control command specified
 */
```

**EXAMPLE 8** No control command specified (Continued)

```
(void) auditwrite(AW_TEXT, "text", AW_END);
```

<b>FILES</b>	/etc/security/audit_event	Used to obtain the mappings between audit event strings and audit event numbers.
--------------	---------------------------	--

<b>Trusted Solaris 8 4/01 Reference Manual</b>	auditconfig(1M), auditreduce(1M), praudit(1M), audit(2), auditon(2), getaudit(2), stat(2), aw_geterrno(3TSOL), aw_perror(3TSOL), aw_perror_r(3TSOL), aw_strerror(3TSOL), audit_event(4)
--	---

*Trusted Solaris Developer's Guide*

<b>SunOS 5.8 Reference Manual</b>	attributes(5)
---------------------------------------	---------------

<b>NOTES</b>	These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.
--------------	---

In a multithreading environment where multiple threads may invoke `auditwrite`, to preserve consistency `AW_USERD` should be used as the first command on all `auditwrite` calls (after a `AW_GETRD` call). Also, `aw_errno` may not be used directly; see `ERRORS` for details.

When a subject is not provided, `auditwrite()` attempts to generate the subject, groups, and (depending on appropriate audit policy) sensitivity label attributes for the current process. This attempt has implications for servers because unless they negotiate to get the AUID, UID, sensitivity 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.

## au\_preselect(3BSM)

NAME	au_preselect – Preselect an audit event										
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;bsm/libbsm.h&gt;  int <b>au_preselect</b>(au_event_t <i>event</i>, au_mask_t *<i>mask_p</i>, int <i>sorf</i>, int                   <i>flag</i>) ;</pre>										
DESCRIPTION	<p><b>au_preselect()</b> determines whether or not the audit event <i>event</i> is preselected against the binary preselection mask pointed to by <i>mask_p</i> (usually obtained by a call to <b>getaudit(2)</b>). <b>au_preselect()</b> looks up the classes associated with <i>event</i> in <b>audit_event(4)</b> 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.</p> <p><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>.</p> <p>The following are the valid values of <i>sorf</i>:</p> <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 failure portions of the preselection mask.</td></tr></table> <p><i>flag</i> tells <b>au_preselect()</b> how to read the <b>audit_event(4)</b> database. Upon initial invocation, <b>au_preselect()</b> reads the <b>audit_event(4)</b> database and allocates space in an internal cache for each entry with <b>malloc(3C)</b>. In subsequent invocations, the value of <i>flag</i> determines where <b>au_preselect()</b> obtains audit event information. The following are the valid values of <i>flag</i>:</p> <table><tr><td>AU_PRS_REREAD</td><td>Get audit event information by searching the <b>audit_event(4)</b> database.</td></tr><tr><td>AU_PRS_USECACHE</td><td>Get audit event information from 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 failure portions of the preselection mask.	AU_PRS_REREAD	Get audit event information by searching the <b>audit_event(4)</b> database.	AU_PRS_USECACHE	Get audit event information from 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 failure portions of the preselection mask.										
AU_PRS_REREAD	Get audit event information by searching the <b>audit_event(4)</b> database.										
AU_PRS_USECACHE	Get audit event information from internal cache created upon the initial invocation. This option is much faster.										
RETURN VALUES	<p><b>au_preselect()</b> returns:</p> <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. <b>au_preselect()</b> couldn't allocate memory or couldn't find <i>event</i> in the <b>audit_event(4)</b> database.</td></tr></table>	0	<i>event</i> is not preselected.	1	<i>event</i> is preselected.	-1	An error occurred. <b>au_preselect()</b> couldn't allocate memory or couldn't find <i>event</i> in the <b>audit_event(4)</b> database.				
0	<i>event</i> is not preselected.										
1	<i>event</i> is preselected.										
-1	An error occurred. <b>au_preselect()</b> couldn't allocate memory or couldn't find <i>event</i> in the <b>audit_event(4)</b> database.										

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
FILES**

This function looks up the classes associated with *event* in the Trusted Solaris *audit\_event(4)* file. By default, auditing is enabled in the Trusted Solaris environment.

<i>/etc/security/audit_class</i>	Maps audit class number to audit class names and descriptions.
<i>/etc/security/audit_event</i>	Maps audit event number to audit event names.

**ATTRIBUTES**

See *attributes(5)* for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual  
NOTES**

*getaudit(2)*, *getauclassent(3BSM)*, *getaeuevent(3BSM)*, *auditwrite(3TSOL)*, *audit\_class(4)*, *audit\_event(4)*

*au\_open(3BSM)*, *malloc(3C)*, *attributes(5)*

This functionality is active only if the auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* 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.

auth\_set\_to\_str(3TSOL)

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
DESCRIPTION	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.



auth\_to\_str(3TSOL)

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
DESCRIPTION	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.

au\_user\_mask(3BSM)

NAME	au_user_mask – Get user’s binary preselection mask					
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int au_user_mask(char *username, au_mask_t *mask_p);</pre>					
DESCRIPTION	<p>au_user_mask() reads the default, system wide audit classes from audit_control(4), combines them with the per-user audit classes from the audit_user(4) database, and updates the binary preselection mask pointed to by mask_p with the combined value.</p> <p>The audit flags in the flags field of the audit_control(4) database and the always-audit-flags and never-audit-flags from the audit_user(4) database represent binary audit classes. These fields are combined by au_preselect(3BSM) as follows:</p> $\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$ <p>au_user_mask() only fails if both the both the audit_control(4) and the audit_user(4) database entries could not be retrieved. This allows for flexible configurations.</p>					
RETURN VALUES	<p>au_user_mask() returns:</p> <table><tr><td>0</td><td>Success.</td></tr><tr><td>-1</td><td>Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.</td></tr></table>		0	Success.	-1	Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.
0	Success.					
-1	Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.					
FILES	<table><tr><td>/etc/security/audit_control</td><td>Contains default parameters read by the audit daemon, auditd(1M).</td></tr><tr><td>/etc/security/audit_user</td><td>Stores per-user audit event mask.</td></tr></table>	/etc/security/audit_control	Contains default parameters read by the audit daemon, auditd(1M).	/etc/security/audit_user	Stores per-user audit event mask.	
/etc/security/audit_control	Contains default parameters read by the audit daemon, auditd(1M).					
/etc/security/audit_user	Stores per-user audit event mask.					
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE					
MT-Level	MT-Safe					
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>By default, auditing is enabled in the Trusted Solaris environment. Trusted Solaris 2.5.1, 7, and later releases extend the number of audit classes and audit events, and introduce new but similar structures and programming interfaces.</p>					
Trusted Solaris 8 4/01 Reference Manual	<p>login(4), getaudit(4), au_preselect(3BSM), getacinfo(3BSM), getausernam(3BSM), audit_control(4), audit_user(4)</p>					
SunOS 5.8 Reference Manual	<p>attributes(5)</p>					

`au_user_mask(3BSM)`

**NOTES** This functionality is active only if auditing has been enabled. `au_user_mask()` should be called by programs like `login(1)` that set the preselection mask of a process with `setaudit(2)` in the Trusted Solaris 8 4/01 Reference Manual. `getaudit(2)` should be used to obtain audit characteristics for the current process.

aw\_errno(3TSOL)

NAME	aw_strerror, aw_errno, aw_geterrno, aw_perror, aw_perror_r – Obtain and display error messages						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file...</i> -lbsm -lsocket -lnsl -lintl -ltsol [<i>library...</i>]  #include &lt;bsm/auditwrite.h&gt; int aw_errno  int <b>aw_geterrno</b>(int <i>rd</i>) ;  char *<b>aw_strerror</b>(const int <i>status</i>) ;  void <b>aw_perror</b>(const char *<i>s</i>) ;  void <b>aw_perror_r</b>(int <i>rd</i>, const char *<i>s</i>) ;</pre>						
DESCRIPTION	<p>aw_errno 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. However, aw_errno cannot be accessed directly in an MT-safe way; in multithreaded environments, aw_geterrno() should be used instead. Supporting functions convert auditwrite(3TSOL) error numbers into text strings.</p> <p>aw_geterrno() takes a record descriptor, <i>rd</i>, and returns the aw_errno value associated with that descriptor.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as "<i>s</i>: <i>error_message</i>" to standard error. This interface is not MT-safe.</p> <p>aw_perror_r() prints the error message corresponding to <i>status</i> associated with the specified record descriptor as "<i>s</i>: <i>error_message</i>".</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWcsu</td></tr><tr><td>MT-Level</td><td>MT-Safe with Exceptions</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe with Exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe with Exceptions						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>auditwrite(3TSOL)</p> <p>errno(3C), perror(3C), strerror(3C), attributes(5)</p> <p>The functions aw_strerror(), aw_geterrno(), and aw_perror_r() are MT-Safe. The aw_errno global variable and the aw_perror() function are not MT-Safe.</p>						

`aw_errno(3TSOL)`

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

aw\_geterrno(3TSOL)

NAME	aw_strerror, aw_errno, aw_geterrno, aw_perror, aw_perror_r – Obtain and display error messages						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file...</i> -lbsm -lsocket -lnsl -lintl -ltsol [<i>library...</i>]  #include &lt;bsm/auditwrite.h&gt; int aw_errno  int <b>aw_geterrno</b>(int <i>rd</i>) ;  char *<b>aw_strerror</b>(const int <i>status</i>) ;  void <b>aw_perror</b>(const char *<i>s</i>) ;  void <b>aw_perror_r</b>(int <i>rd</i>, const char *<i>s</i>) ;</pre>						
DESCRIPTION	<p>aw_errno 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. However, aw_errno cannot be accessed directly in an MT-safe way; in multithreaded environments, aw_geterrno() should be used instead. Supporting functions convert auditwrite(3TSOL) error numbers into text strings.</p> <p>aw_geterrno() takes a record descriptor, <i>rd</i>, and returns the aw_errno value associated with that descriptor.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as "<i>s</i>: <i>error_message</i>" to standard error. This interface is not MT-safe.</p> <p>aw_perror_r() prints the error message corresponding to <i>status</i> associated with the specified record descriptor as "<i>s</i>: <i>error_message</i>".</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWcsu</td></tr><tr><td>MT-Level</td><td>MT-Safe with Exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe with Exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe with Exceptions						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>auditwrite(3TSOL)</p> <p>errno(3C), perror(3C), strerror(3C), attributes(5)</p> <p>The functions aw_strerror(), aw_geterrno(), and aw_perror_r() are MT-Safe. The aw_errno global variable and the aw_perror() function are not MT-Safe.</p>						

`aw_geterrno(3TSOL)`

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

aw\_perror(3TSOL)

NAME	aw_strerror, aw_errno, aw_geterrno, aw_perror, aw_perror_r – Obtain and display error messages						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file...</i> -lbsm -lsocket -lnsl -lintl -ltsol [<i>library...</i>]  #include &lt;bsm/auditwrite.h&gt; int aw_errno  int <b>aw_geterrno</b>(int <i>rd</i>) ;  char *<b>aw_strerror</b>(const int <i>status</i>) ;  void <b>aw_perror</b>(const char *<i>s</i>) ;  void <b>aw_perror_r</b>(int <i>rd</i>, const char *<i>s</i>) ;</pre>						
DESCRIPTION	<p>aw_errno 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. However, aw_errno cannot be accessed directly in an MT-safe way; in multithreaded environments, aw_geterrno() should be used instead. Supporting functions convert auditwrite(3TSOL) error numbers into text strings.</p> <p>aw_geterrno() takes a record descriptor, <i>rd</i>, and returns the aw_errno value associated with that descriptor.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as "<i>s</i>: <i>error_message</i>" to standard error. This interface is not MT-safe.</p> <p>aw_perror_r() prints the error message corresponding to <i>status</i> associated with the specified record descriptor as "<i>s</i>: <i>error_message</i>".</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWcsu</td></tr><tr><td>MT-Level</td><td>MT-Safe with Exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe with Exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe with Exceptions						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>auditwrite(3TSOL)</p> <p>errno(3C), perror(3C), strerror(3C), attributes(5)</p> <p>The functions aw_strerror(), aw_geterrno(), and aw_perror_r() are MT-Safe. The aw_errno global variable and the aw_perror() function are not MT-Safe.</p>						



`aw_perror(3TSOL)`

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

aw\_perror\_r(3TSOL)

NAME	aw_strerror, aw_errno, aw_geterrno, aw_perror, aw_perror_r – Obtain and display error messages						
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl -ltsol [library...]  #include &lt;bsm/auditwrite.h&gt; int aw_errno  int aw_geterrno(int rd);  char *aw_strerror(const int status);  void aw_perror(const char *s);  void aw_perror_r(int rd, const char *s);</pre>						
DESCRIPTION	<p>aw_errno 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. However, aw_errno cannot be accessed directly in an MT-safe way; in multithreaded environments, aw_geterrno() should be used instead. Supporting functions convert auditwrite(3TSOL) error numbers into text strings.</p> <p>aw_geterrno() takes a record descriptor, <i>rd</i>, and returns the aw_errno value associated with that descriptor.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as "s: error_message" to standard error. This interface is not MT-safe.</p> <p>aw_perror_r() prints the error message corresponding to <i>status</i> associated with the specified record descriptor as "s: error_message".</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWcsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with Exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe with Exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe with Exceptions						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>auditwrite(3TSOL)</p> <p>errno(3C), perror(3C), strerror(3C), attributes(5)</p> <p>The functions aw_strerror(), aw_geterrno(), and aw_perror_r() are MT-Safe. The aw_errno global variable and the aw_perror() function are not MT-Safe.</p>						

`aw_perror_r(3TSOL)`

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

aw\_strerror(3TSOL)

NAME	aw_strerror, aw_errno, aw_geterrno, aw_perror, aw_perror_r – Obtain and display error messages						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file...</i> -lbsm -lsocket -lnsl -lintl -ltsol [<i>library...</i>]  #include &lt;bsm/auditwrite.h&gt; int aw_errno  int <b>aw_geterrno</b>(int <i>rd</i>) ;  char *<b>aw_strerror</b>(const int <i>status</i>) ;  void <b>aw_perror</b>(const char *<i>s</i>) ;  void <b>aw_perror_r</b>(int <i>rd</i>, const char *<i>s</i>) ;</pre>						
DESCRIPTION	<p>aw_errno 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. However, aw_errno cannot be accessed directly in an MT-safe way; in multithreaded environments, aw_geterrno() should be used instead. Supporting functions convert auditwrite(3TSOL) error numbers into text strings.</p> <p>aw_geterrno() takes a record descriptor, <i>rd</i>, and returns the aw_errno value associated with that descriptor.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as "<i>s</i>: <i>error_message</i>" to standard error. This interface is not MT-safe.</p> <p>aw_perror_r() prints the error message corresponding to <i>status</i> associated with the specified record descriptor as "<i>s</i>: <i>error_message</i>".</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWcsu</td></tr><tr><td>MT-Level</td><td>MT-Safe with Exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe with Exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe with Exceptions						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>auditwrite(3TSOL)</p> <p>errno(3C), perror(3C), strerror(3C), attributes(5)</p> <p>The functions aw_strerror(), aw_geterrno(), and aw_perror_r() are MT-Safe. The aw_errno global variable and the aw_perror() function are not MT-Safe.</p>						

`aw_strerror(3TSOL)`

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

bclearhigh(3TSOL)

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclabel_t *label) ;</pre>						
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

	bclearhigh(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL), labelvers(3TSOL)
	<i>Trusted Solaris Developer's Guide</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)

bclearlow(3TSOL)

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclear_t *label) ;</pre>						
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						



	bclearlow(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL), labelvers(3TSOL)
	<i>Trusted Solaris Developer's Guide</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)

## bcleartoh(3TSOL)

<b>NAME</b>	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
<b>DESCRIPTION</b>	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
<b>RETURN VALUES</b>	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.
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

bcleartoh(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and bcleartoh\_r() should be used.

## bcleartoh\_r(3TSOL)

<b>NAME</b>	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
<b>DESCRIPTION</b>	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
<b>RETURN VALUES</b>	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.
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

bcleartoh\_r(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions `bcltoh()`, `bsltoh()`, 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()`, and `bcleartoh_r()` should be used.

bclearatos(3TSOL)

NAME	bltos, bcltos, bsltos, bclearatos – translate binary labels to character coded labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int bltos(const blevel_t *label, char **string, const int str_len, const int flags);  int bcltos(const bclabel_t *label, char **string, const int str_len, const int flags);  int bsltos(const bslabel_t *label, char **string, const int str_len, const int flags);  int bclearatos(const bclear_t *label, char **string, const int str_len, const int flags);</pre>						
DESCRIPTION	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <pre>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</pre> <p>Capital letters are used to display all Classification names and Words. The ' ' (space) character separates classifications and words from other words in all character-coded labels except where multiple words that require the same Prefix or Suffix are present, in which case the multiple words are separated from each other by the '/' (slash) character.</p> <p><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.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table><tr><td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i>.</td></tr><tr><td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i>. If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr><tr><td>LONG_CLASSIFICATION</td><td>Translate using long name of classification defined in <i>label</i>.</td></tr></table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.	LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
LONG_WORDS	Translate using long names of words defined in <i>label</i> .						
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.						
LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .						

SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
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 <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

ADMIN\_LOW [ *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).

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] = '\00';).
- >0 If successful, the length of the character-coded label including the NULL terminator.

bclear(3TSOL)

**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 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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL),  
labelinfo(3TSOL)

*Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set*

**SunOS 5.8  
Reference Manual  
NOTES**

free(3C), malloc(3C), attributes(5)

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



NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclabel_t *label) ;</pre>
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

bclearundef(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL),  
labelvers(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

attributes(5)

bclearvalid(3TSOL)

NAME	blvalid, bsvalid, bclearvalid – check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int bsvalid(const bslabel_t *label) ;  int bclearvalid(const bclear_t *clearance) ;</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bsvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
RETURN VALUES	<p>These routines return:</p> <ul style="list-style-type: none"><li>-1        If the label_encodings file is inaccessible.</li><li>0        If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</li><li>1        If the binary label is valid for this system.</li></ul>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p>attributes(5)</p>						
NOTES	<p>Binary sensitivity labels are <i>valid</i> if they are contained in the SYSTEM_ACCREDITATION_RANGE as checked by blinset(3TSOL). bsvalid() is a</p>						

`bclearvalid(3TSOL)`

synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclabel_t *label) ;</pre>
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

bclhigh(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL),  
labelvers(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

attributes(5)

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels						
SYNOPSIS	<pre> <b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *<i>label</i>) ; void <b>bclhigh</b>(bclabel_t *<i>label</i>) ; void <b>bsllow</b>(bsllabel_t *<i>label</i>) ; void <b>bsllhigh</b>(bsllabel_t *<i>label</i>) ; void <b>bclearlow</b>(bclear_t *<i>clearance</i>) ; void <b>bclearhigh</b>(bclear_t *<i>clearance</i>) ; void <b>bclundef</b>(bclabel_t *<i>label</i>) ; void <b>bsllundef</b>(bsllabel_t *<i>label</i>) ; void <b>bclearundef</b>(bclear_t *<i>label</i>) ; </pre>						
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

bcllow(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL),  
labelvers(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

attributes(5)



NAME	bcltobanner – translate binary CMW labels to character-coded labels for a printer banner page
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int <b>bcltobanner</b>(const bclabel_t *label, struct banner_fields *fields,                const int flags);</pre>
DESCRIPTION	<p>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' sensitivity label.</p> <p>bcltobanner() translates a binary CMW label, <i>label</i>, 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.</p> <p>The structure banner_fields stores the following information:</p> <pre>struct banner_fields { char *header;           /* top and bottom banner/trailer page */ char *protect_as;       /* "protect as" banner page section */ char *ilabel;           /* obsolete */ 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 short ilabel_len;  /* obsolete */ short caveats_len;      /* caveats */ short channels_len;     /* handling channels */ };</pre> <p>Members of the <i>fields</i> structure have the following meaning:</p> <p>header String to print at the top and bottom of banner and trailer pages</p> <p>protect_as String to print in the protect as warning of banner and trailer pages</p> <p>ilabel Obsolete string</p> <p>caveats String to print in caveats section of the banner and trailer pages</p>

bcltobanner(3TSOL)

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

Obsolete 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.

0                  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 sensitivity label, 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] = '\00';).

1                  If successful.

## EXAMPLES

### EXAMPLE 1 Banner Page Format

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.

**EXAMPLE 1** Banner Page Format      *(Continued)*

CAVEATS  
CHANNELS  
HEADER

**FILES**      etc/security/tsol/label\_encodings  
            The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**      See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**      blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL),  
                 blportion(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL),  
                 hextob(3TSOL), labelinfo(3TSOL), labelvers(3TSOL), sbltos(3TSOL),  
                 label\_encodings(4)

*Trusted Solaris Developer’s Guide, and Trusted Solaris administrator’s document set*

**SunOS 5.8  
Reference Manual  
NOTES**      free(3C), malloc(3C), attributes(5)

If memory is allocated by this routine, the caller must free memory with `free()` when the memory is no longer in use. ADMIN\_LOW and ADMIN\_HIGH labels are mapped differently than the other routines. If the *label* is ADMIN\_LOW, the *label* is mapped into the minimum sensitivity label defined in the label\_encodings file, and if the *label* is ADMIN\_HIGH, the *label* is mapped into the maximum classification, and all compartments defined in the label\_encodings file.

## bcltoh(3TSOL)

<b>NAME</b>	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
<b>DESCRIPTION</b>	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
<b>RETURN VALUES</b>	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.
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

bcltoh(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and bcleartoh\_r() should be used.

bcltoh\_r(3TSOL)

<b>NAME</b>	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
<b>DESCRIPTION</b>	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
<b>RETURN VALUES</b>	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.
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

bcltoh\_r(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions `bcltoh()`, `bsltoh()`, 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()`, and `bcleartoh_r()` should be used.

## bcltos(3TSOL)

<b>NAME</b>	bltos, bcltos, bsltos, bcleartos – translate binary labels to character coded labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int <b>bltos</b>(const blevel_t *label, char **string, const int str_len, const int flags);  int <b>bcltos</b>(const bclabel_t *label, char **string, const int str_len, const int flags);  int <b>bsltos</b>(const bslabel_t *label, char **string, const int str_len, const int flags);  int <b>bcleartos</b>(const bclear_t *label, char **string, const int str_len, const int flags);</pre>						
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <pre>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</pre> <p>Capital letters are used to display all Classification names and Words. The ' ' (space) character separates classifications and words from other words in all character-coded labels except where multiple words that require the same Prefix or Suffix are present, in which case the multiple words are separated from each other by the '/' (slash) character.</p> <p><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.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i>.</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i>. If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> <tr> <td>LONG_CLASSIFICATION</td><td>Translate using long name of classification defined in <i>label</i>.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.	LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
LONG_WORDS	Translate using long names of words defined in <i>label</i> .						
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.						
LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .						



SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the <i>label_encodings</i> file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name and admin high name strings specified in the <i>label_encodings</i> file. If no strings are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

```
ADMIN_LOW [ 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).

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] = '\00';).
- >0 If successful, the length of the character-coded label including the NULL terminator.

bcltos(3TSOL)

**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 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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL),  
labelinfo(3TSOL)

*Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set*

**SunOS 5.8  
Reference Manual  
NOTES**

free(3C), malloc(3C), attributes(5)

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

NAME	blportion, bcltosl, getcsl, setcsl – access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  bslabel_t *bcltosl(bslabel_t *label);  void getcsl(bslabel_t *destination_label, const bslabel_t *source_label); void setcsl(bslabel_t *destination_label, const bslabel_t *source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() provides a pointer to the sensitivity label of the binary CMW label <i>label</i>.</p> <p>getcsl() copies the sensitivity label of the binary CMW label <i>source_label</i> to the binary sensitivity label <i>destination_label</i>.</p> <p>setcsl() replaces the value of the sensitivity label of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label <i>source_label</i>.</p>						
RETURN VALUES	bcltosl() returns a pointer to its label type.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p><b>EXAMPLE 1</b> Comparing Sensitivity Labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file’s binary sensitivity label.</p> <pre>blequal(bcltosl(&amp;cmw_label), &amp;file_sensitivity_label)</pre>						
Trusted Solaris 8 4/01 Reference Manual	<pre>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</pre> <p><i>Trusted Solaris Developer’s Guide</i></p>						
SunOS 5.8 Reference Manual	<pre>attributes(5)</pre>						

## bclundef(3TSOL)

<b>NAME</b>	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclear_t *label) ;</pre>						
<b>DESCRIPTION</b>	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
<b>ATTRIBUTES</b>	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

	bclundef(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL), labelvers(3TSOL)
	<i>Trusted Solaris Developer's Guide</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)

## bind(3SOCKET)

NAME	bind – bind a name to a socket																						
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lsocket -lnsl [<i>library...</i>]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int <b>bind</b> (int <i>s</i>, const struct sockaddr *<i>name</i>, socklen_t *<i>namelen</i>);</pre>																						
DESCRIPTION	<p><code>bind()</code> assigns a name to an unnamed socket. When a socket is created with <code>socket(3SOCKET)</code>, it exists in a name space (address family) but has no name assigned. <code>bind()</code> requests that the name pointed to by <i>name</i> be assigned to the socket. The socket is bound to a single-level port (SLP), unless the calling process possesses the <code>PRIV_NET_MAC_READ</code> privilege, when it is bound to a multilevel port (MLP).</p>																						
RETURN VALUES	<p><code>bind()</code> returns:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.																		
0	On success.																						
-1	On failure, and sets <code>errno</code> to indicate the error.																						
ERRORS	<p>The <code>bind()</code> call will fail if:</p> <table><tr><td>EACCES</td><td>The requested address is protected and the current user has inadequate permission to access it. The calling process must have the <code>PRIV_NET_PRIVADDR</code> privilege to override this restriction.</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> <p>The following errors are specific to binding names in the UNIX domain:</p> <table><tr><td>EACCES</td><td>Search permission is denied for a component of the path prefix of the pathname 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 pathname was specified.</td></tr></table>	EACCES	The requested address is protected and the current user has inadequate permission to access it. The calling process must have the <code>PRIV_NET_PRIVADDR</code> privilege to override this restriction.	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 pathname in <i>name</i> .	EIO	An I/O error occurred while making the directory entry or allocating the inode.	EISDIR	A null pathname was specified.
EACCES	The requested address is protected and the current user has inadequate permission to access it. The calling process must have the <code>PRIV_NET_PRIVADDR</code> privilege to override this restriction.																						
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 pathname in <i>name</i> .																						
EIO	An I/O error occurred while making the directory entry or allocating the inode.																						
EISDIR	A null pathname was specified.																						

bind(3SOCKET)

ELOOP	Too many symbolic links were encountered in translating the pathname in <i>name</i> .
ENOENT	A component of the path prefix of the pathname in <i>name</i> does not exist.
ENOTDIR	A component of the path prefix of the pathname in <i>name</i> is not a directory.
EROFS	The inode would reside on a read-only file system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

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). To override the access restriction on privileged ports, the calling process must have the PRIV\_NET\_PRIVADDR privilege.

**Trusted Solaris 8  
4/01 Reference  
Manual  
Reference Manual  
NOTES**

unlink(2), socket(3SOCKET)

socket(3HEAD), attributes(5)

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

The rules used in name binding vary between communication domains.

blcompare(3TSOL)

NAME	blcompare, blequal, bldominates, blstrictdom, blinrange – compare binary labels						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt; typedef binary_level_range {     blevel_t lower_bound;     blevel_t upper_bound; } brange_t;  int <b>blequal</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>bldominates</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>blstrictdom</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>blinrange</b>(const blevel_t *<i>label</i>, const brange_t *<i>range</i>);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>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>.</p>						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p><b>EXAMPLE 1</b> Compare two binary labels</p> <p>The following example shows how to compare two binary CMW labels:</p> <pre>blequal(bcltosl(&amp;cmw_label1), bcltosl(&amp;cmw_label2))</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	bcltobanner(3TSOL), bltos(3TSOL), labelinfo(3TSOL) <i>Trusted Solaris Developer's Guide</i>						
SunOS 5.8 Reference Manual	attributes(5)						



<b>NAME</b>	bcompare, blequal, bldominates, blstrictdom, blinrange – compare binary labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt; 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 bldominates(const blevel_t *label1, const blevel_t *label2); int blstrictdom(const blevel_t *label1, const blevel_t *label2); int blinrange(const blevel_t *label, const brange_t *range);</pre>						
<b>DESCRIPTION</b>	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>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>.</p>						
<b>RETURN VALUES</b>	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Compare two binary labels</p> <p>The following example shows how to compare two binary CMW labels:</p> <pre>blequal(bcltosl(&amp;cmw_label1), bcltosl(&amp;cmw_label2))</pre>						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), bltos(3TSOL), labelinfo(3TSOL) <i>Trusted Solaris Developer's Guide</i>						
<b>SunOS 5.8 Reference Manual</b>	attributes(5)						

blequal(3TSOL)

NAME	blcompare, blequal, bldominates, blstrictdom, blinrange – compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt; 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 bldominates(const blevel_t *label1, const blevel_t *label2); int blstrictdom(const blevel_t *label1, const blevel_t *label2); int blinrange(const blevel_t *label, const brange_t *range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>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>.</p>						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p><b>EXAMPLE 1</b> Compare two binary labels</p> <p>The following example shows how to compare two binary CMW labels:</p> <pre>blequal(bcltosl(&amp;cmw_label1), bcltosl(&amp;cmw_label2))</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	bcltobanner(3TSOL), bltos(3TSOL), labelinfo(3TSOL) <i>Trusted Solaris Developer's Guide</i>						
SunOS 5.8 Reference Manual	attributes(5)						

<b>NAME</b>	blcompare, blequal, bldominates, blstrictdom, blinrange – compare binary labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt; typedef binary_level_range {     blevel_t lower_bound;     blevel_t upper_bound; } brange_t;  int <b>blequal</b>(const blevel_t *label1, const blevel_t *label2); int <b>bldominates</b>(const blevel_t *label1, const blevel_t *label2); int <b>blstrictdom</b>(const blevel_t *label1, const blevel_t *label2); int <b>blinrange</b>(const blevel_t *label, const brange_t *range);</pre>						
<b>DESCRIPTION</b>	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>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>.</p>						
<b>RETURN VALUES</b>	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Compare two binary labels</p> <p>The following example shows how to compare two binary CMW labels:</p> <pre>blequal(bcltosl(&amp;cmw_label1), bcltosl(&amp;cmw_label2))</pre>						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), bltos(3TSOL), labelinfo(3TSOL) <i>Trusted Solaris Developer's Guide</i>						
<b>SunOS 5.8 Reference Manual</b>	attributes(5)						

blinset(3TSOL)

NAME	blinset – Check binary label for inclusion in set						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt; typedef label_set_identifier {     int type;     char *name; } set_id;  int <b>blinset</b>(const bslabel_t *<i>label</i>, const set_id *<i>id</i>);</pre>						
DESCRIPTION	<p>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.</p> <p><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:</p> <p>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.</p> <p>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.</p> <p>Other <i>type</i> and <i>name</i> values are reserved for future implementation.</p>						
RETURN VALUES	<p>blinset() returns:</p> <ul style="list-style-type: none"><li>1            If the label is contained in the specified set.</li><li>0            If the label is not in the specified set, is not a valid sensitivity label, or is not dominated by the process sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges.</li><li>-1           If the specified set is inaccessible.</li></ul>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

	blinset(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), labelinfo(3TSOL), sbltos(3TSOL), label_encodings(4)
	<i>Trusted Solaris Developer's Guide and the Trusted Solaris administrator's document set</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)
<b>NOTES</b>	The ADMIN_HIGH and ADMIN_LOW labels are accepted even if the remainder of the system's accreditation range is inaccessible.
<b>BUGS</b>	The only sets available are the System Accreditation Range and User Accreditation Range.

## blmanifest(3TSOL)

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bcllabel_t *<i>label</i>) ; void <b>bclhigh</b>(bcllabel_t *<i>label</i>) ; void <b>bsllow</b>(bsllabel_t *<i>label</i>) ; void <b>bsllhigh</b>(bsllabel_t *<i>label</i>) ; void <b>bclearlow</b>(bclear_t *<i>clearance</i>) ; void <b>bclearhigh</b>(bclear_t *<i>clearance</i>) ; void <b>bclundef</b>(bcllabel_t *<i>label</i>) ; void <b>bsllundef</b>(bsllabel_t *<i>label</i>) ; void <b>bclearundef</b>(bclear_t *<i>label</i>) ;</pre>						
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef() and bsllundef() initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

	blmanifest(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL), labelvers(3TSOL)
	<i>Trusted Solaris Developer's Guide</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)

blmaximum(3TSOL)

NAME	blminmax, blmaximum, blminimum – Bound of two binary levels						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>blmaximum</b>(blevel_t *maximum_label, const blevel_t                *bounding_label) ;  void <b>blminimum</b>(blevel_t *minimum_label, const blevel_t *bounding_label) ;</pre>						
DESCRIPTION	<p><b>blmaximum()</b> replaces the contents of binary level <i>maximum_label</i> with the least upper bound of binary levels <i>maximum_label</i> and <i>bounding_label</i>. The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.</p> <p><b>blminimum()</b> replaces the contents of binary level <i>minimum_label</i> with the greatest lower bound of binary levels <i>minimum_label</i> and <i>bounding_label</i>. The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.</p>						
ATTRIBUTES	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p><a href="#">bcltobanner(3TSOL)</a>, <a href="#">blinset(3TSOL)</a>, <a href="#">blvalid(3TSOL)</a>, <a href="#">labelinfo(3TSOL)</a>, <a href="#">sbltos(3TSOL)</a></p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p><a href="#">attributes(5)</a></p>						



NAME	blminmax, blmaximum, blminimum – Bound of two binary levels						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>blmaximum</b>(blevel_t *maximum_label, const blevel_t                *bounding_label) ;  void <b>blminimum</b>(blevel_t *minimum_label, const blevel_t *bounding_label) ;</pre>						
DESCRIPTION	<p><b>blmaximum()</b> replaces the contents of binary level <i>maximum_label</i> with the least upper bound of binary levels <i>maximum_label</i> and <i>bounding_label</i>. The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.</p> <p><b>blminimum()</b> replaces the contents of binary level <i>minimum_label</i> with the greatest lower bound of binary levels <i>minimum_label</i> and <i>bounding_label</i>. The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.</p>						
ATTRIBUTES	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p><a href="#">bcltobanner(3TSOL)</a>, <a href="#">blinset(3TSOL)</a>, <a href="#">blvalid(3TSOL)</a>, <a href="#">labelinfo(3TSOL)</a>, <a href="#">sbltos(3TSOL)</a></p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p><a href="#">attributes(5)</a></p>						

blminmax(3TSOL)

NAME	blminmax, blmaximum, blminimum – Bound of two binary levels						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void blmaximum(blevel_t *maximum_label, const blevel_t                *bounding_label) ;  void blminimum(blevel_t *minimum_label, const blevel_t *bounding_label) ;</pre>						
DESCRIPTION	<p>blmaximum() replaces the contents of binary level <i>maximum_label</i> with the least upper bound of binary levels <i>maximum_label</i> and <i>bounding_label</i>. The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.</p> <p>blminimum() replaces the contents of binary level <i>minimum_label</i> with the greatest lower bound of binary levels <i>minimum_label</i> and <i>bounding_label</i>. The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>bcltobanner(3TSOL), blinset(3TSOL), blvalid(3TSOL), labelinfo(3TSOL), sbltos(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p>attributes(5)</p>						

NAME	blportion, bcltosl, getcsl, setcsl – access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  bslabel_t *bcltosl(bslabel_t *label);  void getcsl(bslabel_t *destination_label, const bslabel_t *source_label); void setcsl(bslabel_t *destination_label, const bslabel_t *source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() provides a pointer to the sensitivity label of the binary CMW label <i>label</i>.</p> <p>getcsl() copies the sensitivity label of the binary CMW label <i>source_label</i> to the binary sensitivity label <i>destination_label</i>.</p> <p>setcsl() replaces the value of the sensitivity label of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label <i>source_label</i>.</p>						
RETURN VALUES	bcltosl() returns a pointer to its label type.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p><b>EXAMPLE 1</b> Comparing Sensitivity Labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file’s binary sensitivity label.</p> <pre>blequal(bcltosl(&amp;cmw_label), &amp;file_sensitivity_label)</pre>						
Trusted Solaris 8 4/01 Reference Manual	<pre>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</pre> <p><i>Trusted Solaris Developer’s Guide</i></p>						
SunOS 5.8 Reference Manual	<pre>attributes(5)</pre>						

blstrictdom(3TSOL)

NAME	blcompare, blequal, bldominates, blstrictdom, blinrange – compare binary labels						
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt; typedef binary_level_range {     blevel_t lower_bound;     blevel_t upper_bound; } brange_t;  int <b>blequal</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>bldominates</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>blstrictdom</b>(const blevel_t *<i>label1</i>, const blevel_t *<i>label2</i>); int <b>blinrange</b>(const blevel_t *<i>label</i>, const brange_t *<i>range</i>);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>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>.</p>						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p><b>EXAMPLE 1</b> Compare two binary labels</p> <p>The following example shows how to compare two binary CMW labels:</p> <pre>blequal(bcltosl(&amp;cmw_label1), bcltosl(&amp;cmw_label2))</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	bcltobanner(3TSOL), bltos(3TSOL), labelinfo(3TSOL) <i>Trusted Solaris Developer's Guide</i>						
SunOS 5.8 Reference Manual	attributes(5)						

<b>NAME</b>	bltcolor, bltcolor_r – Get character-coded color name of label						
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bltcolor(const blevel_t *label);  char *bltcolor_r(const blevel_t *label, const int size, char                 *color_name);</pre>						
<b>DESCRIPTION</b>	<p>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.</p> <p>bltcolor() and bltcolor_r() get the character-coded color name associated with the binary label <i>label</i>.</p>						
<b>RETURN VALUES</b>	<p>bltcolor() returns a pointer to a statically allocated string that contains the character-coded color name specified for the <i>label</i> or returns (char *) 0 if, for any reason, no character-coded color name is available for this binary label.</p> <p>bltcolor_r() returns a pointer to the <i>color_name</i> string which contains the character-coded color name specified for the <i>label</i> or returns (char *) 0 if, for any reason, no character-coded color name is available for this binary label. <i>color_name</i> must provide for a string of at least <i>size</i> characters.</p>						
<b>FILES</b>	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe with exceptions						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide, Trusted Solaris User's Guide, and Trusted Solaris administrator's document set</i></p>						
<b>SunOS 5.8 Reference Manual NOTES</b>	<p>attributes(5)</p> <p>The function bltcolor() returns a pointer to a statically allocated string. Subsequent calls to it will overwrite that string with a new character-coded color name. It is not MT-Safe.</p>						

bltcolor(3TSOL)

For multithreaded applications the function `bltcolor_r()` should be used.

If *label* includes a specified word or words, the character-coded color name associated with the first word specified in the label encodings file is returned. Otherwise, if no character-coded color name is specified for *label*, the first character-coded color name specified in the label encodings file with the same classification as the binary label is returned.

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

<b>NAME</b>	bltocolor, bltocolor_r – Get character-coded color name of label						
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bltocolor(const blevel_t *label);  char *bltocolor_r(const blevel_t *label, const int size, char                  *color_name);</pre>						
<b>DESCRIPTION</b>	<p>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.</p> <p>bltocolor() and bltocolor_r() get the character-coded color name associated with the binary label <i>label</i>.</p>						
<b>RETURN VALUES</b>	<p>bltocolor() returns a pointer to a statically allocated string that contains the character-coded color name specified for the <i>label</i> or returns (char *) 0 if, for any reason, no character-coded color name is available for this binary label.</p> <p>bltocolor_r() returns a pointer to the <i>color_name</i> string which contains the character-coded color name specified for the <i>label</i> or returns (char *) 0 if, for any reason, no character-coded color name is available for this binary label. <i>color_name</i> must provide for a string of at least <i>size</i> characters.</p>						
<b>FILES</b>	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe with exceptions						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide, Trusted Solaris User's Guide, and Trusted Solaris administrator's document set</i></p>						
<b>SunOS 5.8 Reference Manual NOTES</b>	<p>attributes(5)</p> <p>The function bltocolor() returns a pointer to a statically allocated string. Subsequent calls to it will overwrite that string with a new character-coded color name. It is not MT-Safe.</p>						

bltcolor\_r(3TSOL)

For multithreaded applications the function `bltcolor_r()` should be used.

If *label* includes a specified word or words, the character-coded color name associated with the first word specified in the label encodings file is returned. Otherwise, if no character-coded color name is specified for *label*, the first character-coded color name specified in the label encodings file with the same classification as the binary label is returned.

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



<b>NAME</b>	bltos, bcltos, bsltos, bcleartos – translate binary labels to character coded labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltso1 [library...]  #include &lt;tsol/label.h&gt;  int <b>bltos</b>(const blevel_t *label, char **string, const int str_len, const int flags);  int <b>bcltos</b>(const bclabel_t *label, char **string, const int str_len, const int flags);  int <b>bsltos</b>(const bslabel_t *label, char **string, const int str_len, const int flags);  int <b>bcleartos</b>(const bclear_t *label, char **string, const int str_len, const int flags);</pre>						
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <pre>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</pre> <p>Capital letters are used to display all Classification names and Words. The ' ' (space) character separates classifications and words from other words in all character-coded labels except where multiple words that require the same Prefix or Suffix are present, in which case the multiple words are separated from each other by the '/' (slash) character.</p> <p><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.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i>.</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i>. If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> <tr> <td>LONG_CLASSIFICATION</td><td>Translate using long name of classification defined in <i>label</i>.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.	LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
LONG_WORDS	Translate using long names of words defined in <i>label</i> .						
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.						
LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .						

bltos(3TSOL)

SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the <i>label_encodings</i> file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name and admin high name strings specified in the <i>label_encodings</i> file. If no strings are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

ADMIN\_LOW [ *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).

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 <i>label_encodings</i> 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]='\\00';).  |
| >0 | If successful, the length of the character-coded label including the NULL terminator.   |

bltos(3TSOL)

**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 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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolo(3TSOL),  
labelinfo(3TSOL)

*Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set*

**SunOS 5.8  
Reference Manual**

free(3C), malloc(3C), attributes(5)

**NOTES**

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

bltype(3TSOL)

NAME	bltype, setbltype – compare and set the type of binary label										
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int <b>bltype</b>(const void *label, const unsigned char type); void <b>setbltype</b>(void *label, const unsigned char type);</pre>										
DESCRIPTION	<p>These functions compare and set the type of binary labels.</p> <p><code>bltype()</code> examines <i>label</i> to determine if it is of the specified type <i>type</i>.</p> <p><code>setbltype()</code> sets the type of <i>label</i> to the specified type <i>type</i>.</p> <p><i>type</i> may be one of:</p> <table><tr><td>SUN_SL_ID</td><td><i>label</i> is a defined binary sensitivity label.</td></tr><tr><td>SUN_SL_UN</td><td><i>label</i> is an undefined binary sensitivity label.</td></tr><tr><td>SUN_CLR_ID</td><td><i>label</i> is a defined binary clearance.</td></tr><tr><td>SUN_CLR_UN</td><td><i>label</i> is an undefined binary clearance.</td></tr><tr><td>SUN_CMW_ID</td><td><i>label</i> is a binary CMW label whose label portions may or may not be defined. (<code>bltype()</code> only.)</td></tr></table>	SUN_SL_ID	<i>label</i> is a defined binary sensitivity label.	SUN_SL_UN	<i>label</i> is an undefined binary sensitivity label.	SUN_CLR_ID	<i>label</i> is a defined binary clearance.	SUN_CLR_UN	<i>label</i> is an undefined binary clearance.	SUN_CMW_ID	<i>label</i> is a binary CMW label whose label portions may or may not be defined. ( <code>bltype()</code> only.)
SUN_SL_ID	<i>label</i> is a defined binary sensitivity label.										
SUN_SL_UN	<i>label</i> is an undefined binary sensitivity label.										
SUN_CLR_ID	<i>label</i> is a defined binary clearance.										
SUN_CLR_UN	<i>label</i> is an undefined binary clearance.										
SUN_CMW_ID	<i>label</i> is a binary CMW label whose label portions may or may not be defined. ( <code>bltype()</code> only.)										
RETURN VALUES	<code>bltype()</code> returns non-zero if <i>label</i> is of type <i>type</i> , otherwise zero is returned.										
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Availability	SUNWtsu										
MT-Level	MT-Safe										
Trusted Solaris 8 4/01 Reference Manual	<p><code>bcltobanner(3TSOL)</code>, <code>blcompare(3TSOL)</code>, <code>bltocolour(3TSOL)</code>, <code>btohex(3TSOL)</code>, <code>labelinfo(3TSOL)</code></p> <p><i>Trusted Solaris Developer's Guide</i></p>										
SunOS 5.8 Reference Manual WARNINGS	<p><code>attributes(5)</code></p> <ol style="list-style-type: none"><li><code>bltype(&amp;cmw_label, SUN_CMW_ID)</code> checks the existence of a binary CMW label structure and not the portions of the structure that contain defined labels.</li><li>When attempting to determine the type of a label, rather than to verify that a specific label type is present, check <code>SUN_CMW_ID</code> first.</li><li><code>setbltype()</code> makes no checks on the structure it is setting or the type value.</li></ol>										

blvalid(3TSOL)

NAME	blvalid, bsvalid, bclearvalid – check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int bsvalid(const bslabel_t *label) ;  int bclearvalid(const bclear_t *clearance) ;</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bsvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
RETURN VALUES	<p>These routines return:</p> <ul style="list-style-type: none"><li>-1        If the label_encodings file is inaccessible.</li><li>0        If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</li><li>1        If the binary label is valid for this system.</li></ul>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p>attributes(5)</p>						
NOTES	<p>Binary sensitivity labels are <i>valid</i> if they are contained in the SYSTEM_ACCREDITATION_RANGE as checked by blinset(3TSOL). bsvalid() is a</p>						

`blvalid(3TSOL)`

synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclabel_t *label) ;</pre>
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

bslhigh(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL),  
labelvers(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

attributes(5)



NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bsllundef, bclearundef – create manifest binary labels
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bsllabel_t *label) ; void <b>bsllhigh</b>(bsllabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bsllundef</b>(bsllabel_t *label) ; void <b>bclearundef</b>(bclabel_t *label) ;</pre>
DESCRIPTION	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bsllundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

bsllow(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL),  
labelvers(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

attributes(5)

NAME	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
SYNOPSIS	<pre> <b>cc</b> [<i>flag...</i>] <i>file</i> ... -ltsol [<i>library...</i>]  #include &lt;tsol/label.h&gt;  char *<b>bcltoh</b>(const bclabel_t *<i>label</i>) ; char *<b>bsltoh</b>(const bslabel_t *<i>label</i>) ; char *<b>bcleartoh</b>(const bclear_t *<i>clearance</i>) ; char *<b>bcltoh_r</b>(const bclabel_t *<i>label</i>, char *<i>hex</i>) ; char *<b>bsltoh_r</b>(const bslabel_t *<i>label</i>, char *<i>hex</i>) ; char *<b>bcleartoh_r</b>(const bclear_t *<i>clearance</i>, char *<i>hex</i>) ; char *<b>h_alloc</b>(const unsigned char <i>type</i>) ; void <b>h_free</b>(char *<i>hex</i>) ; </pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre> SUN_CMW_ID      <i>label</i> is a binary CMW label. SUN_SL_ID       <i>label</i> is a binary sensitivity label. SUN_CLR_ID      <i>label</i> is a binary clearance. </pre> <p>h_free() frees memory allocated by h_alloc().</p>
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.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

bsltoh(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and  
bcleartoh\_r() should be used.

NAME	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
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.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

bsltoh\_r(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and bcleartoh\_r() should be used.

<b>NAME</b>	bltos, bcltos, bsltos, bcleartos – translate binary labels to character coded labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltso1 [library...]  #include &lt;tso1/label.h&gt;  int <b>bltos</b>(const blevel_t *label, char **string, const int str_len, const int flags);  int <b>bcltos</b>(const bclabel_t *label, char **string, const int str_len, const int flags);  int <b>bsltos</b>(const bslabel_t *label, char **string, const int str_len, const int flags);  int <b>bcleartos</b>(const bclear_t *label, char **string, const int str_len, const int flags);</pre>						
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <pre>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</pre> <p>Capital letters are used to display all Classification names and Words. The ' ' (space) character separates classifications and words from other words in all character-coded labels except where multiple words that require the same Prefix or Suffix are present, in which case the multiple words are separated from each other by the '/' (slash) character.</p> <p><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.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i>.</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i>. If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> <tr> <td>LONG_CLASSIFICATION</td><td>Translate using long name of classification defined in <i>label</i>.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.	LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
LONG_WORDS	Translate using long names of words defined in <i>label</i> .						
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.						
LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .						

bsltos(3TSOL)

SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the <i>label_encodings</i> file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name and admin high name strings specified in the <i>label_encodings</i> file. If no strings are specified, the strings "ADMIN_LOW" and "ADMIN_HIGH" are used.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

ADMIN\_LOW [ *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).

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 <i>label_encodings</i> 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]='\\00';).
>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 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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolo(3TSOL),  
labelinfo(3TSOL)

*Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set*

**SunOS 5.8  
Reference Manual**

free(3C), malloc(3C), attributes(5)

**NOTES**

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

## bslundef(3TSOL)

<b>NAME</b>	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, bclearlow, bclearhigh, bclundef, bslundef, bclearundef – create manifest binary labels						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  void <b>bcllow</b>(bclabel_t *label) ; void <b>bclhigh</b>(bclabel_t *label) ; void <b>bsllow</b>(bslabel_t *label) ; void <b>bsllhigh</b>(bslabel_t *label) ; void <b>bclearlow</b>(bclear_t *clearance) ; void <b>bclearhigh</b>(bclear_t *clearance) ; void <b>bclundef</b>(bclabel_t *label) ; void <b>bslundef</b>(bslabel_t *label) ; void <b>bclearundef</b>(bclear_t *label) ;</pre>						
<b>DESCRIPTION</b>	<p>These functions initialize binary label structures to manifest values.</p> <p><b>bcllow()</b> and <b>bclhigh()</b> initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p><b>bsllow()</b> and <b>bsllhigh()</b> initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p><b>bclearlow()</b> and <b>bclearhigh()</b> initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p><b>bclundef()</b> and <b>bslundef()</b> initialize the binary CMW and sensitivity label structure <i>label</i> to the manifest constant value for an undefined CMW and sensitivity label, respectively.</p> <p><b>bclearundef()</b> initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
<b>ATTRIBUTES</b>	<p>See <a href="#">attributes(5)</a> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

	bslundef(3TSOL)
<b>Trusted Solaris 8 4/01 Reference Manual</b>	bcltobanner(3TSOL), blcompare(3TSOL), bltype(3TSOL), hex tob(3TSOL), labelvers(3TSOL)
	<i>Trusted Solaris Developer's Guide</i>
<b>SunOS 5.8 Reference Manual</b>	attributes(5)

bslvalid(3TSOL)

NAME	blvalid, bslvalid, bclearvalid – check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int bslvalid(const bslabel_t *label) ;  int bclearvalid(const bclear_t *clearance) ;</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
RETURN VALUES	<p>These routines return:</p> <table><tr><td>-1</td><td>If the label_encodings file is inaccessible.</td></tr><tr><td>0</td><td>If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</td></tr><tr><td>1</td><td>If the binary label is valid for this system.</td></tr></table>	-1	If the label_encodings file is inaccessible.	0	If the binary label is not valid for this system or is not dominated by the process' sensitivity label 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.
-1	If the label_encodings file is inaccessible.						
0	If the binary label is not valid for this system or is not dominated by the process' sensitivity label 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	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p>attributes(5)</p>						
NOTES	<p>Binary sensitivity labels are <i>valid</i> if they are contained in the SYSTEM_ACCREDITATION_RANGE as checked by blinset(3TSOL). bslvalid() is a</p>						

bslvalid(3TSOL)

synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

## btohex(3TSOL)

<b>NAME</b>	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
<b>DESCRIPTION</b>	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
<b>RETURN VALUES</b>	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.
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

btohex(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions `bcltoh()`, `bsltoh()`, 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()`, and `bcleartoh_r()` should be used.

## chkauth(3TSOL)

<b>NAME</b>	chkauth – Verify user authorizations
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The <code>chkauth</code> function is replaced in Trusted Solaris 8 and later releases with the <code>chkauthattr</code> function described in the <code>getauthattr(3SECDB)</code> man page. This function finds authorization entries in <code>auth_attr(4)</code> .



NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void); authattr_t *getauthnam(const char *name); void free_authattr(authattr_t *auth); void setauthattr(void); void endauthattr(void); int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The <code>getauthattr()</code> and <code>getauthnam()</code> functions each return an <code>auth_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getauthattr()</code> function enumerates <code>auth_attr</code> entries. The <code>getauthnam()</code> function searches for an <code>auth_attr</code> entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive <code>auth_attr</code> entries or NULL.</p> <p>The internal representation of an <code>auth_attr</code> entry is an <code>authattr_t</code> structure defined in <code>&lt;auth_attr.h&gt;</code> with the following members:</p> <pre>char  *name;           /* name of the authorization */ char  *res1;           /* reserved for future use */ char  *res2;           /* reserved for future use */ char  *short_desc;     /* short description */ char  *long_desc;      /* long description */ kva_t *attr;           /* array of key-value pair attributes */</pre> <p>The <code>setauthattr()</code> function “rewinds” to the beginning of the enumeration of <code>auth_attr</code> entries. Calls to <code>getauthnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setauthattr()</code> should be called before the first call to <code>getauthattr()</code>.</p> <p>The <code>endauthattr()</code> function may be called to indicate that <code>auth_attr</code> processing is complete; the system may then close any open <code>auth_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>chkauthattr()</code> function verifies whether or not a user has a given authorization. It first reads the <code>AUTHS_GRANTED</code> key in the <code>/etc/security/policy.conf</code> file and returns 1 if it finds a match for the given authorization. If <code>chkauthattr()</code> does not find a match, it reads the <code>PROFS_GRANTED</code> key in <code>/etc/security/policy.conf</code> and returns 1 if the given authorization is in any profiles specified with the <code>PROFS_GRANTED</code> keyword. If a</p>

## chkauthattr(3SECDB)

match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	/etc/security/policy.conf <b>or</b>	<b>Is user</b>
Authorization name	<code>user_attr</code> <b>or</b> <code>prof_attr</code> <b>entry</b>	<b>authorized?</b>
solaris.printer.postscript	solaris.printer.postscript	Yes
solaris.printer.postscript	solaris.printer.*	Yes
solaris.printer.grant	solaris.printer.*	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

### RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

### USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.



## clnt\_call(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

clnt\_call(3NSL)

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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcrerror()` [see `rpc_clnt_create(3NSL)`], `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*);

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const rpcproc\_t *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 *nettype* is NULL, it 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 it returns with appropriate status.

## clnt\_call(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `tfattr_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 `tfalloc_blk()` routine to allocate

clnt\_call(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`,  
`t6free_blk(3NSL)`

`printf(3C)`, `rpc_clnt_auth(3NSL)`, `attributes(5)`

## clnt\_control(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS  rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS  rpcvers_t    set the RPC program's version                                    number associated with the</pre>



client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t      get program number
CLSET_PROG     rpcprog_t      set program number
```

The following 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 and FALSE on failure.

```
CLIENT *clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t
versnum, const char *nettype) ;
```

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype) ;
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_control(3NSL)

routine is successful it 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(3NSL)*). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

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

A function macro that 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 *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(3NSL)*).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)*). 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.

clnt\_control(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_control(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_control(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_create(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR    struct netbuf * get server's address CLGET_FD          int *          get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                 number associated with the                                 client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                 number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

```
CLIENT *clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t
versnum, const char *nettype) ;
```

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

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

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype) ;
```

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.



clnt\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_create\_timed(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>

clnt\_create\_timed(3NSL)

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

CLIENT \*clnt\_create(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype) ;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

CLIENT \*clnt\_create\_timed(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype, const struct timeval \*timeout);

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype) ;

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_create\_timed(3NSL)

routine is successful it 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(3NSL)`). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

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

A function macro that 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 `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(3NSL)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)`). 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.

clnt\_create\_timed(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_create\_timed(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.



clnt\_create\_timed(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

clnt\_create\_vers(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS  rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS  rpcvers_t    set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

CLIENT \*clnt\_create(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype) ;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

CLIENT \*clnt\_create\_timed(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype, const struct timeval \*timeout);

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype) ;

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_create\_vers(3NSL)

routine is successful it 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(3NSL)`). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

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

A function macro that 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 `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(3NSL)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)`). 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.

clnt\_create\_vers(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_create\_vers(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_create\_vers(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

**SunOS 5.8  
Reference Manual**

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_create\_vers\_timed(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>



## clnt\_create\_vers\_timed(3NSL)

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t      get program number
CLSET_PROG     rpcprog_t      set program number
```

The following 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 and FALSE on failure.

CLIENT \*clnt\_create(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype) ;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

CLIENT \*clnt\_create\_timed(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype, const struct timeval \*timeout);

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, const char \*nettype) ;

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_create\_vers\_timed(3NSL)

routine is successful it 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(3NSL)*). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

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

A function macro that 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 *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(3NSL)*).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)*). 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.

clnt\_create\_vers\_timed(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_create\_vers\_timed(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

`clnt_create_vers_timed(3NSL)`

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_destroy(3NSL)

<b>NAME</b>	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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</p>
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_destroy(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.



clnt\_destroy(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_destroy(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_destroy(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_dg\_create(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS        rpcvers_t          get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS        rpcvers_t          set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID    uint32_t    get the XID of the previous
                remote procedure call
CLSET_XID    uint32_t    set the XID of the next
                remote procedure call
CLGET_PROG   rpcprog_t   get program number
CLSET_PROG   rpcprog_t   set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_dg\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.

clnt\_dg\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_dg\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.



clnt\_dg\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

clnt\_freeres(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_freeres(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

## SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

clnt\_freeres(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpc(3NSL), rpc_clnt_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL),  
t6free_blk(3NSL)`

`printf(3C), rpc_clnt_auth(3NSL), attributes(5)`

clnt\_geterr(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);  A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);  A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);  A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcrerror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_geterr(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `tfattr_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 `tfalloc_blk()` routine to allocate



clnt\_geterr(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpc(3NSL), rpc_clnt_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL),  
t6free_blk(3NSL)`

`printf(3C), rpc_clnt_auth(3NSL), attributes(5)`

clnt\_pcreateerror(3NSL)

<b>NAME</b>	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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</p>
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS        rpcvers_t          get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS        rpcvers_t          set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_pcreateerror(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.

clnt\_pcreateerror(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_pcreateerror(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

	clnt_pcreateerror(3NSL)
	When <code>clnt_destroy()</code> is used to destroy a client handle, the client should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that client handle.
Trusted Solaris 8 4/01 Reference Manual	<code>rpcbind(1M)</code> , <code>rpc(3NSL)</code> , <code>rpc_clnt_calls(3NSL)</code> , <code>rpc_svc_create(3NSL)</code> , <code>libt6(3NSL)</code> , <code>t6alloc_blk(3NSL)</code> , <code>t6free_blk(3NSL)</code>
SunOS 5.8 Reference Manual	<code>rpc_clnt_auth(3NSL)</code> , <code>svc_raw_create(3NSL)</code> , <code>attributes(5)</code>

clnt\_perrno(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p><code>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</code> A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p><code>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</code> A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p><code>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</code> A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>



```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_perrno(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

clnt\_perino(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`rpc(3NSL), rpc_clnt_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL),  
t6free_blk(3NSL)`

**SunOS 5.8  
Reference Manual**

`printf(3C), rpc_clnt_auth(3NSL), attributes(5)`

clnt\_perror(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p><code>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</code> A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p><code>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</code> A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p><code>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</code> A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcrerror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_perror(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `tfattr_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 `tfalloc_blk()` routine to allocate

clnt\_perror(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`,  
`t6free_blk(3NSL)`

`printf(3C)`, `rpc_clnt_auth(3NSL)`, `attributes(5)`

clnt\_raw\_create(3NSL)

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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</p>
DESCRIPTION	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR    struct netbuf * get server's address CLGET_FD          int *          get associated file descriptor 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</pre> <pre>CLGET_VERS  rpcvers_t      get the RPC program's version                                 number associated with the                                 client handle CLSET_VERS  rpcvers_t      set the RPC program's version                                 number associated with the</pre>



client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

```
CLIENT *clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t
versnum, const char *nettype) ;
```

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

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

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype) ;
```

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_raw\_create(3NSL)

routine is successful it 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(3NSL)`). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 `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(3NSL)`).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)`). 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.

clnt\_raw\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)*). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC\_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is RPC\_ANYFD, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is RPC\_ANYFD and *netconf* is NULL, a RPC\_UNKNOWNPROTO error is set. If *fildev* 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(1M)*) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

clnt\_raw\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*, const struct timeval \**timeout*);

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 *fildev*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 *fildev* is an open and bound file descriptor. The user may specify 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.

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;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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.

clnt\_raw\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_spcreateerror(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>

clnt\_spcreateerror(3NSL)

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

CLIENT \*clnt\_create(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype) ;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

CLIENT \*clnt\_create\_timed(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype, const struct timeval \*timeout);

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, const char \*nettype) ;

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_spcreateerror(3NSL)

routine is successful it 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(3NSL)`). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 `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(3NSL)`).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)`). 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.



clnt\_spcreateerror(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_spcreateerror(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_spcrcreateerror(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_sperrno(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <a href="#">SUMMARY OF TRUSTED SOLARIS CHANGES</a> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

clnt\_sperrno(3NSL)

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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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*);

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const rpcproc\_t *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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_sperrno(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

	clnt_sperrno(3NSL)
	attribute-control structures and set the <code>t6attr_t</code> pointers in the <code>CLIENT</code> structure. When <code>clnt_destroy()</code> is used to destroy a client handle, the client should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that client handle.
Trusted Solaris 8 4/01 Reference Manual	<code>rpc(3NSL)</code> , <code>rpc_clnt_create(3NSL)</code> , <code>libt6(3NSL)</code> , <code>t6alloc_blk(3NSL)</code> , <code>t6free_blk(3NSL)</code>
SunOS 5.8 Reference Manual	<code>printf(3C)</code> , <code>rpc_clnt_auth(3NSL)</code> , <code>attributes(5)</code>

## clnt\_sperror(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>



## clnt\_sperror(3NSL)

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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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*);

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const rpcproc\_t *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 *nettype* is NULL, it 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 it returns with appropriate status.

clnt\_sperror(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

clnt\_sperror(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`,  
`t6free_blk(3NSL)`

**SunOS 5.8  
Reference Manual**

`printf(3C)`, `rpc_clnt_auth(3NSL)`, `attributes(5)`

## clnt\_tli\_create(3NSL)

<b>NAME</b>	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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</p>
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

```
CLIENT *clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t
versnum, const char *nettype) ;
```

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

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

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype) ;
```

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_tli\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.

clnt\_tli\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_tli\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.



clnt\_tli\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## clnt\_tp\_create(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS        rpcvers_t          get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS        rpcvers_t          set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID    uint32_t    get the XID of the previous
                remote procedure call
CLSET_XID    uint32_t    set the XID of the next
                remote procedure call
CLGET_PROG   rpcprog_t   get program number
CLSET_PROG   rpcprog_t   set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_tp\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.

clnt\_tp\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_tp\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_tp\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

clnt\_tp\_create\_timed(3NSL)

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, 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
DESCRIPTION	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS        rpcvers_t          get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS        rpcvers_t          set the RPC program's version                                    number associated with the</pre>



## clnt\_tp\_create\_timed(3NSL)

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

CLIENT \*clnt\_create(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype) ;

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)).

CLIENT \*clnt\_create\_timed(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const char \*nettype, const struct timeval \*timeout);

Generic client creation routine which is similar to clnt\_create() but which also 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 rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype) ;

Generic client creation routine which is similar to clnt\_create() but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_tp\_create\_timed(3NSL)

routine is successful it 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(3NSL)`). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

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

A function macro that 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 `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(3NSL)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)`). 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.

clnt\_tp\_create\_timed(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

clnt\_tp\_create\_timed(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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.

clnt\_tp\_create\_timed(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

**SunOS 5.8  
Reference Manual**

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

clnt\_vc\_create(3NSL)

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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</p>
DESCRIPTION	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS  rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS  rpcvers_t    set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t     get program number
CLSET_PROG     rpcprog_t     set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## clnt\_vc\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.



clnt\_vc\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## clnt\_vc\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

clnt\_vc\_create(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

clock\_getres(3RT)

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations				
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lrt [<i>library...</i>]  #include &lt;time.h&gt;  int <b>clock_settime</b>(clockid_t <i>clock_id</i>, const struct timespec *<i>tp</i>); int <b>clock_gettime</b>(clockid_t <i>clock_id</i>, struct timespec *<i>tp</i>); int <b>clock_getres</b>(clockid_t <i>clock_id</i>, struct timespec *<i>res</i>);</pre>				
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. 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. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <i>tp</i> for the specified clock, <i>clock_id</i>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <i>res</i> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <i>res</i>. If <i>res</i> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <i>res</i>, then the value is truncated to a multiple of <i>res</i>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process).</p> <p>A <i>clock_id</i> of <code>CLOCK_REALTIME</code> is defined in <code>&lt;time.h&gt;</code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch.</p> <p>A <i>clock_id</i> of <code>CLOCK_HIGHRES</code> represents the non-adjustable, high-resolution clock for the system. For this clock, the value returned by <code>clock_gettime()</code> represents the amount of time (in seconds and nanoseconds) since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is not subject to resetting or drifting by way of <code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>settimeofday(3C)</code>, or <code>clock_settime()</code>. The time source for this clock is the same as that for <code>gethrtime(3C)</code>.</p> <p>Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>				
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.
0	On success.				
-1	On failure, and sets <code>errno</code> to indicate the error.				

	clock_getres(3RT)				
<b>ERRORS</b>	<p>The <code>clock_gettime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <p><b>EINVAL</b>            The <i>clock_id</i> argument does not specify a known clock.</p> <p><b>ENOSYS</b>            The functions <code>clock_gettime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.</p> <p>The <code>clock_gettime()</code> function will fail if:</p> <p><b>EINVAL</b>            The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.</p> <p>The <code>clock_gettime()</code> function may fail if:</p> <p><b>EPERM</b>            The requesting process does not have the appropriate privilege to set the specified clock.</p>				
<b>ATTRIBUTES</b>	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td><code>clock_gettime()</code> is Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe				
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b> SunOS 5.8 Reference Manual	<p>The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p><code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>time(2)</code>, <code>ctime(3C)</code>, <code>gethrtime(3C)</code>, <code>settimeofday(3C)</code>, <code>time(3HEAD)</code>, <code>timer_gettime(3RT)</code>, <code>attributes(5)</code></p>				

## clock\_gettime(3RT)

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations				
SYNOPSIS	<pre>cc [flags...] file ... -lrt [library...]  #include &lt;time.h&gt;  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);</pre>				
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. 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. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <i>tp</i> for the specified clock, <i>clock_id</i>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <i>res</i> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <i>res</i>. If <i>res</i> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <i>res</i>, then the value is truncated to a multiple of <i>res</i>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process).</p> <p>A <i>clock_id</i> of <code>CLOCK_REALTIME</code> is defined in <code>&lt;time.h&gt;</code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch.</p> <p>A <i>clock_id</i> of <code>CLOCK_HIGHRES</code> represents the non-adjustable, high-resolution clock for the system. For this clock, the value returned by <code>clock_gettime()</code> represents the amount of time (in seconds and nanoseconds) since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is not subject to resetting or drifting by way of <code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>settimeofday(3C)</code>, or <code>clock_settime()</code>. The time source for this clock is the same as that for <code>gethrtime(3C)</code>.</p> <p>Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>				
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.
0	On success.				
-1	On failure, and sets <code>errno</code> to indicate the error.				

	clock_gettime(3RT)								
<b>ERRORS</b>	<p>The <code>clock_gettime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <table> <tr> <td>EINVAL</td><td>The <i>clock_id</i> argument does not specify a known clock.</td></tr> <tr> <td>ENOSYS</td><td>The functions <code>clock_gettime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.</td></tr> </table> <p>The <code>clock_gettime()</code> function will fail if:</p> <table> <tr> <td>EINVAL</td><td>The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.</td></tr> </table> <p>The <code>clock_gettime()</code> function may fail if:</p> <table> <tr> <td>EPERM</td><td>The requesting process does not have the appropriate privilege to set the specified clock.</td></tr> </table>	EINVAL	The <i>clock_id</i> argument does not specify a known clock.	ENOSYS	The functions <code>clock_gettime()</code> , <code>clock_gettime()</code> , and <code>clock_getres()</code> are not supported by this implementation.	EINVAL	The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.	EPERM	The requesting process does not have the appropriate privilege to set the specified clock.
EINVAL	The <i>clock_id</i> argument does not specify a known clock.								
ENOSYS	The functions <code>clock_gettime()</code> , <code>clock_gettime()</code> , and <code>clock_getres()</code> are not supported by this implementation.								
EINVAL	The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.								
EPERM	The requesting process does not have the appropriate privilege to set the specified clock.								
<b>ATTRIBUTES</b>	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td><code>clock_gettime()</code> is Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe								
<b>SUMMARY OF TRUSTED SOLARIS CHANGES SunOS 5.8 Reference Manual</b>	<p>The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p><code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>time(2)</code>, <code>ctime(3C)</code>, <code>gethrtime(3C)</code>, <code>settimeofday(3C)</code>, <code>time(3HEAD)</code>, <code>timer_gettime(3RT)</code>, <code>attributes(5)</code></p>								

## clock\_settime(3RT)

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations				
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lrt [<i>library...</i>]  #include &lt;time.h&gt;  int <b>clock_settime</b>(clockid_t <i>clock_id</i>, const struct timespec *<i>tp</i>); int <b>clock_gettime</b>(clockid_t <i>clock_id</i>, struct timespec *<i>tp</i>); int <b>clock_getres</b>(clockid_t <i>clock_id</i>, struct timespec *<i>res</i>);</pre>				
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. 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. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <i>tp</i> for the specified clock, <i>clock_id</i>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <i>res</i> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <i>res</i>. If <i>res</i> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <i>res</i>, then the value is truncated to a multiple of <i>res</i>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process).</p> <p>A <i>clock_id</i> of <code>CLOCK_REALTIME</code> is defined in <code>&lt;time.h&gt;</code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch.</p> <p>A <i>clock_id</i> of <code>CLOCK_HIGHRES</code> represents the non-adjustable, high-resolution clock for the system. For this clock, the value returned by <code>clock_gettime()</code> represents the amount of time (in seconds and nanoseconds) since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is not subject to resetting or drifting by way of <code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>settimeofday(3C)</code>, or <code>clock_settime()</code>. The time source for this clock is the same as that for <code>gethrtime(3C)</code>.</p> <p>Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>				
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.
0	On success.				
-1	On failure, and sets <code>errno</code> to indicate the error.				



	clock_gettime(3RT)								
<b>ERRORS</b>	<p>The <code>clock_gettime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <table> <tr> <td>EINVAL</td><td>The <i>clock_id</i> argument does not specify a known clock.</td></tr> <tr> <td>ENOSYS</td><td>The functions <code>clock_gettime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.</td></tr> </table> <p>The <code>clock_gettime()</code> function will fail if:</p> <table> <tr> <td>EINVAL</td><td>The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.</td></tr> </table> <p>The <code>clock_gettime()</code> function may fail if:</p> <table> <tr> <td>EPERM</td><td>The requesting process does not have the appropriate privilege to set the specified clock.</td></tr> </table>	EINVAL	The <i>clock_id</i> argument does not specify a known clock.	ENOSYS	The functions <code>clock_gettime()</code> , <code>clock_gettime()</code> , and <code>clock_getres()</code> are not supported by this implementation.	EINVAL	The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.	EPERM	The requesting process does not have the appropriate privilege to set the specified clock.
EINVAL	The <i>clock_id</i> argument does not specify a known clock.								
ENOSYS	The functions <code>clock_gettime()</code> , <code>clock_gettime()</code> , and <code>clock_getres()</code> are not supported by this implementation.								
EINVAL	The <i>tp</i> argument to <code>clock_gettime()</code> is outside the range for the given clock ID; or the <i>tp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.								
EPERM	The requesting process does not have the appropriate privilege to set the specified clock.								
<b>ATTRIBUTES</b>	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td><code>clock_gettime()</code> is Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe								
<b>SUMMARY OF TRUSTED SOLARIS CHANGES SunOS 5.8 Reference Manual</b>	<p>The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p><code>adjtime(2)</code>, <code>ntp_adjtime(2)</code>, <code>time(2)</code>, <code>ctime(3C)</code>, <code>gethrtime(3C)</code>, <code>settimeofday(3C)</code>, <code>time(3HEAD)</code>, <code>timer_gettime(3RT)</code>, <code>attributes(5)</code></p>								

## dn\_comp(3RESOLV)

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket  -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

```

int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);

```

**DESCRIPTION**

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_STAYOPEN</code>	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
<code>RES_IGNTC</code>	Ignore truncation errors; that is, do not retry with TCP.

## dn\_comp(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

## dn\_comp(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the <code>HOSTALIASES</code> environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, <code>NULL</code> is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or <code>-1</code> if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is <code>NULL</code>, names are not compressed. If <i>lastdnptr</i> is <code>NULL</code>, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or <code>-1</code> if there was an error.
<b>hstrerror, herror</b>	<p>The variables <code>statp-&gt;res_h_errno</code> and <code>_res.res_h_errno</code> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

# SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

## Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

## NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

## dn\_expand(3RESOLV)

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herron – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket  -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>



dn\_expand(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
             anslen);

void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

## dn\_expand(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

## dn\_expand(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the <code>HOSTALIASES</code> environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, <code>NULL</code> is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or <code>-1</code> if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is <code>NULL</code>, names are not compressed. If <i>lastdnptr</i> is <code>NULL</code>, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or <code>-1</code> if there was an error.
<b>hstrerror, herror</b>	<p>The variables <code>statp-&gt;res_h_errno</code> and <code>_res.res_h_errno</code> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See <code>attributes(5)</code> for descriptions of the following attributes:

dn\_expand(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

# SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

## Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

## NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

## door\_create(3DOOR)

NAME	door_create – Create a door descriptor						
SYNOPSIS	<pre>cc [flags...] file ... -ldoor -lthread [library...]  #include &lt;door.h&gt;  int door_create(void (*server_procedure) (void *cookie, void *cookie, char     *argp, size_t arg_size, door_desc_t *dp, uint_t n_desc, void     *cookie, uint_t attributes);</pre>						
DESCRIPTION	<p>The <code>door_create()</code> function creates a door descriptor that describes the procedure specified by the function <code>server_procedure</code>.</p> <p>In the Trusted Solaris environment, <code>door_create()</code> sets the door CMW label and MAC policy based on attributes of the creating process. The CMW label of the door is set equal to the CMW label of the calling process. If the calling process has the <code>sys_system_door</code> privilege, the MAC policy is set to <code>MAC_ANY</code>, otherwise its MAC policy is set to <code>MAC_EQUAL</code>.</p> <p>The data item, <code>cookie</code>, is associated with the door descriptor, and is passed as an argument to the invoked function <code>server_procedure</code> during <code>door_call(3X)</code> invocations. Other arguments passed to <code>server_procedure</code> from an associated <code>door_call()</code> are placed on the stack and include <code>argp</code> and <code>dp</code>. <code>argp</code> points to <code>arg_size</code> bytes of data and <code>dp</code> points to <code>n_desc</code> <code>door_desc_t</code> structures. The <code>attributes</code> flag specifies attributes associated with the newly created door. Valid values for <code>attributes</code> are constructed by ORing in one or more of the following values:</p> <table> <tr> <td>DOOR_UNREF</td><td>Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code>. In the case of an unreferenced invocation, the values for <code>arg_size</code>, <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.</td></tr> <tr> <td>DOOR_UNREF_MULTI</td><td>Similar to <code>DOOR_UNREF</code>, except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.</td></tr> <tr> <td>DOOR_PRIVATE</td><td>Maintains a separate pool of server threads on behalf of the door. Server threads are associated with a door's private server pool using <code>door_bind(3X)</code>.</td></tr> </table>	DOOR_UNREF	Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code> . In the case of an unreferenced invocation, the values for <code>arg_size</code> , <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.	DOOR_UNREF_MULTI	Similar to <code>DOOR_UNREF</code> , except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.	DOOR_PRIVATE	Maintains a separate pool of server threads on behalf of the door. Server threads are associated with a door's private server pool using <code>door_bind(3X)</code> .
DOOR_UNREF	Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code> . In the case of an unreferenced invocation, the values for <code>arg_size</code> , <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.						
DOOR_UNREF_MULTI	Similar to <code>DOOR_UNREF</code> , except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.						
DOOR_PRIVATE	Maintains a separate pool of server threads on behalf of the door. Server threads are associated with a door's private server pool using <code>door_bind(3X)</code> .						

door\_create(3DOOR)

The descriptor returned from `door_create()` will be marked as close on exec (`FD_CLOEXEC`). Information about a door is available for all clients of a door using `door_info(3DOOR)`. Programs concerned with security should not place secure information in door data that is accessible by `door_info()`. In particular, secure data should not be stored in the data item *cookie*.

By default, additional threads are created as needed to handle concurrent `door_call(3DOOR)` invocations. See `door_server_create(3DOOR)` for information on how to change this behavior.

#### RETURN VALUES

`door_create()` returns:

>0            On success.

-1            On failure, and sets `errno` to indicate the error.

#### ERRORS

The `door_create()` function will fail if:

`EINVAL`            Invalid attributes are passed.

`EMFILE`            The process has too many open descriptors.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	all
Availability	SUNWcsu
Stability	Evolving
MT-Level	Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

In the Trusted Solaris environment, `door_create()` sets the door CMW label and MAC policy based on attributes of the creating process. The CMW label of the door is set equal to the CMW label of the calling process. If the calling process has the `sys_system_door` privilege, the MAC policy is set to `MAC_ANY`, otherwise its MAC policy is set to `MAC_EQUAL`.

#### Trusted Solaris 8 4/01 Reference Manual

`door_tcred(3DOOR)`

`door_bind(3DOOR)`, `door_call(3DOOR)`, `door_info(3DOOR)`,  
`door_revoke(3DOOR)`, `door_server_create(3DOOR)`, `attributes(5)`

## door\_tcred(3DOOR)

<b>NAME</b>	door_tcred – Return the extended credential information associated with the client of the current door invocation										
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -ltsol -lthread [library...]  #include &lt;door.h&gt; #include &lt;sys/tsol/tdoor.h&gt;  int door_tcred(door_tcred_t *info);</pre>										
<b>DESCRIPTION</b>	<p>The door_tcred() function returns the extended credential information associated with the client (if any) of the current door invocation.</p> <p>The tsol_door_cred_t structure is returned by the door_tcred() interface. It is added to the Trusted Solaris environment so that a door server is able to get the Trusted Solaris attributes of the calling client.</p> <pre>/*  * Structure used to return info from door_tcred  */ typedef struct tsol_door_cred {     door_cred_t    tdc_cred;        /* cred data */     bclabel_t      tdc_cmw_label;   /* CMW Label */     bclear_t       tdc_clearance;   /* Clearance */     pattr_t        tdc_proc_attr;   /* Proc. Attr. Flags */     priv_set_t     tdc_effective;   /* Effective set */ } tsol_door_cred_t;</pre> <p>The credential information associated with the client refers to the information from the immediate caller, not necessarily from the first thread in a chain of door calls.</p>										
<b>RETURN VALUES</b>	<p>door_tcred() returns:</p> <p>0            On success.</p> <p>–1           On failure, and sets errno to indicate the error.</p>										
<b>ERRORS</b>	<p>The door_tcred() function fails if:</p> <p>EFAULT            The address of the <i>info</i> argument is invalid.</p> <p>EINVAL            There is no associated door client.</p>										
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Architecture</td><td>all</td></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>Stability</td><td>Unstable</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Architecture	all	Availability	SUNWtsu	Stability	Unstable	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Architecture	all										
Availability	SUNWtsu										
Stability	Unstable										
MT-Level	Safe										



door\_tcred(3DOOR)

door\_create(3DOOR)

door\_call(3DOOR), door\_cred(3DOOR), attributes(5)

It would be more appropriate to use an extensible mechanism rather than the door\_tcred() call. This is expected to be part of the general extension mechanism for process attributes, and will be addressed then. The current door\_tcred() can be re-implemented in terms of such a general mechanism.

endac(3BSM)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information	
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>ns</code> -l<code>intl</code> [<i>library...</i>]  #include &lt;bsm/libbsm.h&gt;  int <b>getacdir</b>( char *<i>dir</i>, int <i>len</i> ); int <b>getacmin</b>( int *<i>min_val</i> ); int <b>getacflg</b>( char *<i>auditstring</i>, int <i>len</i> ); int <b>getacna</b>( char *<i>auditstring</i>, int <i>len</i> ); void <b>setac</b>( void ); void <b>endac</b>( void );</pre>	
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> 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 <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>	
FILES	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0           on success.</p> <p>–2          On failure and set <code>errno</code> to indicate the error.</p> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <p>1           On EOF.</p> <p><code>getacdir()</code> returns:</p>	

endac(3BSM)

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 If the directory entry format in the `audit_control` file is incorrect.
- `getacdir()`, `getacflg()` and `getacna()` return:
- 3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

endauclass(3BSM)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_class_ent *<b>getauclassnam</b>( const char *<i>name</i> );  struct au_class_ent *<b>getauclassnam_r</b>( au_class_ent_t *<i>class_int</i>,     const char *<i>name</i> );  struct au_class_ent *<b>getauclassent</b>( void );  struct au_class_ent *<b>getauclassent_r</b>( au_class_ent_t *<i>class_int</i> );  void <b>setauclass</b>( void );  void <b>endauclass</b>( void );</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre>char      *ac_name; au_class_t  ac_class; char      *ac_desc;</pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

endauclass(3BSM)

getauclassent() and getauclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES** /etc/security/audit\_class Maps audit class numbers to audit class names.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
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.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

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.

endauevent(3BSM)

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endauevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>

RETURN VALUES

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

- `/etc/security/audit_event` Maps audit event numbers to audit event names.
- `/etc/passwd` Stores user-ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
Trusted Solaris 8  
4/01 Reference  
Manual  
Trusted Solaris  
Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

- `getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`
- `getpwnam(3C)`, `passwd(4)`, `attributes(5)`

endauevent(3BSM)

<b>NOTES</b>	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.
--------------	---



NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void); authattr_t *getauthnam(const char *name); void free_authattr(authattr_t *auth); void setauthattr(void); void endauthattr(void); int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The <code>getauthattr()</code> and <code>getauthnam()</code> functions each return an <code>auth_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getauthattr()</code> function enumerates <code>auth_attr</code> entries. The <code>getauthnam()</code> function searches for an <code>auth_attr</code> entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive <code>auth_attr</code> entries or NULL.</p> <p>The internal representation of an <code>auth_attr</code> entry is an <code>authattr_t</code> structure defined in <code>&lt;auth_attr.h&gt;</code> with the following members:</p> <pre>char *name;           /* name of the authorization */ char *res1;           /* reserved for future use */ char *res2;           /* reserved for future use */ char *short_desc;     /* short description */ char *long_desc;      /* long description */ kva_t *attr;          /* array of key-value pair attributes */</pre> <p>The <code>setauthattr()</code> function “rewinds” to the beginning of the enumeration of <code>auth_attr</code> entries. Calls to <code>getauthnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setauthattr()</code> should be called before the first call to <code>getauthattr()</code>.</p> <p>The <code>endauthattr()</code> function may be called to indicate that <code>auth_attr</code> processing is complete; the system may then close any open <code>auth_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>chkauthattr()</code> function verifies whether or not a user has a given authorization. It first reads the <code>AUTHS_GRANTED</code> key in the <code>/etc/security/policy.conf</code> file and returns 1 if it finds a match for the given authorization. If <code>chkauthattr()</code> does not find a match, it reads the <code>PROFS_GRANTED</code> key in <code>/etc/security/policy.conf</code> and returns 1 if the given authorization is in any profiles specified with the <code>PROFS_GRANTED</code> keyword. If a</p>

endauthattr(3SECDB)

match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	/etc/security/policy.conf or	Is user
Authorization name	user_attr or prof_attr entry	authorized?
solaris.printer.postscript	solaris.printer.postscript	Yes
solaris.printer.postscript	solaris.printer.*	Yes
solaris.printer.grant	solaris.printer.*	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

#### RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

#### USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

	endauthattr(3SECDB)				
	Individual attributes in the attr structure can be referred to by calling the kva_match(3SECDB) function.				
<b>WARNINGS</b>	Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.				
<b>FILES</b>	<div> <div>/etc/nsswitch.conf</div> <div>configuration file lookup information for the name server switch</div> </div> <div> <div>/etc/user_attr</div> <div>extended user attributes</div> </div> <div> <div>/etc/security/auth_attr</div> <div>authorization attributes</div> </div> <div> <div>/etc/security/policy.conf</div> <div>policy definitions</div> </div> <div> <div>/etc/security/prof_attr</div> <div>profile information</div> </div>				
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b> Trusted Solaris 8 4/01 Reference Manual SunOS 5.10 Reference Manual	<p>The Trusted Solaris environment adds authorizations. The chkauthattr() function replaces the Trusted Solaris 7 chkauth() function.</p> <p>nsswitch.conf(4), prof_attr(4), user_attr(4)</p> <p>getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB), auth_attr(4), attributes(5), rbac(5)</p>				

endauuser(3BSM)

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre><b>cc</b> [<i>flag...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt;  #include &lt;bsm/libbsm.h&gt;  struct au_user_ent *<b>getauusernam</b>(const char *<i>name</i>); struct au_user_ent *<b>getauuserent</b>(void); void <b>setauuser</b>(void); void <b>endauuser</b>(void);  struct au_user_ent *<b>getauusernam_r</b>(au_user_ent_t * <i>u</i>, const char     *<i>name</i>);  struct au_user_ent *<b>getauuserent_r</b>(au_user_ent_t *<i>u</i>);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry. Entries can come from any of the sources specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or NULL.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code>&lt;bsm/libbsm.h&gt;</code> with the following members:</p> <pre>char      *au_name; au_mask_t au_always; au_mask_t au_never;</pre>

endauser(3BSM)

## RETURN VALUES

The `getauusernam()` function returns a pointer to a `struct_au_user_ent` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `getauuserent()` function returns a pointer to a `struct_au_user_ent` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

## FILES

`/etc/security/audit_user`

Stores per-user audit event mask.

`/etc/passwd`

Stores user-id to username mappings.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual NOTES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_user(4)`, `nsswitch.conf(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

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

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

endexecattr(3SECDB)

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or NULL. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to NULL.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                 /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>

The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See **EXAMPLES**.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

## RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and

endexecattr(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

## EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (\*), if defined, is returned.

```
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

## FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe



endexecattr(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

endprofattr(3SECDB)

NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;prof.h&gt;  profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	<p>The getprofattr() and getprofnam() functions each return a prof_attr entry. Entries can come from any of the sources specified in the nsswitch.conf(4) file.</p> <p>The getprofattr() function enumerates prof_attr entries. The getprofnam() function searches for a prof_attr entry with a given <i>name</i>. Successive calls to these functions return either successive prof_attr entries or NULL.</p> <p>The internal representation of a prof_attr entry is a profattr_t structure defined in &lt;prof_attr.h&gt; with the following members:</p> <pre>char    name;    /* Name of the profile */ char    res1;    /* Reserved for future use */ char    res2;    /* Reserved for future use */ char    desc;    /* Description/Purpose of the profile */ kva_t   attr;    /* Profile attributes */</pre> <p>The free_profattr() function releases memory allocated by the getprofattr() and getprofnam() functions.</p> <p>The setprofattr() function “rewinds” to the beginning of the enumeration of prof_attr entries. Calls to getprofnam() can leave the enumeration in an indeterminate state. Therefore, setprofattr() should be called before the first call to getprofattr().</p> <p>The endprofattr() function may be called to indicate that prof_attr processing is complete; the system may then close any open prof_attr file, deallocate storage, and so forth.</p> <p>The getproflist() function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by getproflist() should be freed using the free_proflist() function.</p>

endprofattr(3SECDB)

	<p>The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.</p>				
RETURN VALUES	<p>The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>				
USAGE	<p>Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function.</p> <p>Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error.</p> <p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function.</p> <p>Applications 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. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.</p>				
FILES	<p><code>/etc/security/prof_attr</code> profiles and their descriptions</p>				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<p><code>auths(1)</code>, <code>profiles(1)</code>, <code>getexecattr(3SECDB)</code>, <code>getauthattr(3SECDB)</code>, <code>prof_attr(4)</code></p>				

endprofent(3TSOL)

<b>NAME</b>	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get user profile description
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getprofent, setprofent, endprofent, getprofentbyname, and free_profent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

endprofstr(3TSOL)

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] ( <b>obsolete</b> )
DESCRIPTION	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

## enduserattr(3SECDB)

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ flag... ] file... -lsecdb -lsocket -lnsl -lintl [ library... ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>

enduserattr(3SECDB)

**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/user_attr</code>	extended user attributes
<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

enduserent(3TSOL)

<b>NAME</b>	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).



NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];      /* user login name */ char          ut_id[4];       /* /sbin/inittab id (usually line #) */ char          ut_line[12];    /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;           /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

endutent(3C)

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

	endutent(3C)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	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.
<b>SunOS 5.8 Reference Manual</b>	ttyslot(3C), utmp(4), utmpx(4), attributes(5)
<b>NOTES</b>	<p>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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().</p>

## endutxent(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## endutxent(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the <code>e_termination</code> and <code>e_exit</code> members of the <code>ut_exit</code> structure are valid only for records of type <code>DEAD_PROCESS</code>. For utmpx entries created by <code>init(1M)</code>, these values are set according to the result of the <code>wait()</code> call that <code>init</code> performs on the process when the process exits. See the <code>wait(2)</code> manual page for the values <code>init</code> uses. Applications creating utmpx entries can set <code>ut_exit</code> values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See <code>wstat(3XFN)</code> for descriptions of the <code>WTERMSIG</code> and <code>WEXITSTATUS</code> macros.</p> <p>The <code>ut_session</code> member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type <code>USER_PROCESS</code>, the <code>nonuser()</code> and <code>nonuserx()</code> macros use the value of the <code>ut_exit.e_exit</code> member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each <code>pty</code> to have a utmpx record (as most applications expect.). The <code>NONROOT_USER</code> macro defines the value that <code>login</code> places in the <code>ut_exit.e_exit</code> member.</p>
RETURN VALUES	<p>Upon successful completion, <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to <code>getutxid()</code> or <code>getutxline()</code>.</p> <p>Upon successful completion, <code>pututxline()</code> returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference**

**Manual**

**Reference Manual**

**NOTES**

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

fp\_resstat(3RESOLV)

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket  -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>



```

int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);

```

**DESCRIPTION**

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_STAYOPEN</code>	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
<code>RES_IGNTC</code>	Ignore truncation errors; that is, do not retry with TCP.

## fp\_resstat(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

fp\_resstat(3RESOLV)

<b>res_hostalias</b>	<p>The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the <code>HOSTALIASES</code> environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, <code>NULL</code> is returned. <code>res_hostalias()</code> stores the result in <i>buf</i>.</p>
<b>res_nclose</b>	<p>The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i>.</p>
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or <code>-1</code> if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is <code>NULL</code>, names are not compressed. If <i>lastdnptr</i> is <code>NULL</code>, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	<p>The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i>, which is of size <i>length</i>. <code>dn_expand()</code> returns the size of the compressed name, or <code>-1</code> if there was an error.</p>
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<p><code>/etc/resolv.conf</code> Resolver configuration file</p>
<b>ATTRIBUTES</b>	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p>

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

# SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

## Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

## NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

## free\_authattr(3SECDB)

NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void); authattr_t *getauthnam(const char *name); void free_authattr(authattr_t *auth); void setauthattr(void); void endauthattr(void); int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The getauthattr() and getauthnam() functions each return an auth_attr(4) entry. Entries can come from any of the sources specified in the nsswitch.conf(4) file.</p> <p>The getauthattr() function enumerates auth_attr entries. The getauthnam() function searches for an auth_attr entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive auth_attr entries or NULL.</p> <p>The internal representation of an auth_attr entry is an authattr_t structure defined in &lt;auth_attr.h&gt; with the following members:</p> <pre>char  *name;           /* name of the authorization */ char  *res1;           /* reserved for future use */ char  *res2;           /* reserved for future use */ char  *short_desc;     /* short description */ char  *long_desc;      /* long description */ kva_t *attr;           /* array of key-value pair attributes */</pre> <p>The setauthattr() function “rewinds” to the beginning of the enumeration of auth_attr entries. Calls to getauthnam() can leave the enumeration in an indeterminate state. Therefore, setauthattr() should be called before the first call to getauthattr().</p> <p>The endauthattr() function may be called to indicate that auth_attr processing is complete; the system may then close any open auth_attr file, deallocate storage, and so forth.</p> <p>The chkauthattr() function verifies whether or not a user has a given authorization. It first reads the AUTHS_GRANTED key in the /etc/security/policy.conf file and returns 1 if it finds a match for the given authorization. If chkauthattr() does not find a match, it reads the PROFS_GRANTED key in /etc/security/policy.conf and returns 1 if the given authorization is in any profiles specified with the PROFS_GRANTED keyword. If a</p>

free\_authattr(3SECDB)

match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	/etc/security/policy.conf or	Is user
Authorization name	user_attr or prof_attr entry	authorized?
solaris.printer.postscript	solaris.printer.postscript	Yes
solaris.printer.postscript	solaris.printer.*	Yes
solaris.printer.grant	solaris.printer.*	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

free\_authattr(3SECDB)

Individual attributes in the `attr` structure can be referred to by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/auth_attr</code>	authorization attributes
<code>/etc/security/policy.conf</code>	policy definitions
<code>/etc/security/prof_attr</code>	profile information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF TRUSTED SOLARIS CHANGES**  
Trusted Solaris 8 4/01 Reference Manual  
The Trusted Solaris environment adds authorizations. The `chkauthattr()` function replaces the Trusted Solaris 7 `chkauth()` function.

`nsswitch.conf(4)`, `prof_attr(4)`, `user_attr(4)`

`getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `getuserattr(3SECDB)`, `kva_match(3SECDB)`, `auth_attr(4)`, `attributes(5)`, `rbac(5)`



	free_auth_set(3TSOL)
<b>NAME</b>	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ]  ( <b>obsolete</b> )
<b>DESCRIPTION</b>	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.

free\_execattr(3SECDB)

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or NULL. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to NULL.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                   /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>

free\_execattr(3SECDB)

The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See **EXAMPLES**.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

## RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and

free\_execattr(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

## EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (\*), if defined, is returned.

```
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

## FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

`free_execattr(3SECDB)`

**SEE ALSO** `getauthattr(3SECDB)`, `getuserattr(3SECDB)`, `kva_match(3SECDB)`,  
`exec_attr(4)`, `policy.conf(4)`, `user_attr(4)`, `attributes(5)`

free\_profattr(3SECDB)

NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;prof.h&gt;  profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	<p>The getprofattr() and getprofnam() functions each return a prof_attr entry. Entries can come from any of the sources specified in the nsswitch.conf(4) file.</p> <p>The getprofattr() function enumerates prof_attr entries. The getprofnam() function searches for a prof_attr entry with a given <i>name</i>. Successive calls to these functions return either successive prof_attr entries or NULL.</p> <p>The internal representation of a prof_attr entry is a profattr_t structure defined in &lt;prof_attr.h&gt; with the following members:</p> <pre>char    name;    /* Name of the profile */ char    res1;    /* Reserved for future use */ char    res2;    /* Reserved for future use */ char    desc;    /* Description/Purpose of the profile */ kva_t   attr;    /* Profile attributes */</pre> <p>The free_profattr() function releases memory allocated by the getprofattr() and getprofnam() functions.</p> <p>The setprofattr() function “rewinds” to the beginning of the enumeration of prof_attr entries. Calls to getprofnam() can leave the enumeration in an indeterminate state. Therefore, setprofattr() should be called before the first call to getprofattr().</p> <p>The endprofattr() function may be called to indicate that prof_attr processing is complete; the system may then close any open prof_attr file, deallocate storage, and so forth.</p> <p>The getproflist() function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by getproflist() should be freed using the free_proflist() function.</p>

free\_profattr(3SECDB)

	<p>The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.</p>				
RETURN VALUES	<p>The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>				
USAGE	<p>Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function.</p> <p>Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error.</p> <p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function.</p> <p>Applications 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. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.</p>				
FILES	<p><code>/etc/security/prof_attr</code> profiles and their descriptions</p>				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<p><code>auths(1)</code>, <code>profiles(1)</code>, <code>getexecattr(3SECDB)</code>, <code>getauthattr(3SECDB)</code>, <code>prof_attr(4)</code></p>				

free\_profent(3TSOL)

<b>NAME</b>	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get user profile description
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getprofent, setprofent, endprofent, getprofentbyname, and free_profent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).



free\_profstr(3TSOL)

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] ( <b>obsolete</b> )
DESCRIPTION	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

free\_userattr(3SECDB)

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ flag... ] file... - lsecdb - lsocket - lns1 - lint1 [ library... ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>

free\_userattr(3SECDB)

**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/user_attr</code>	extended user attributes
<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

free\_userent(3TSOL)

<b>NAME</b>	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).

NAME	ftw, nftw – Walk a file tree																		
SYNOPSIS	<pre>#include &lt;ftw.h&gt;  int <b>ftw</b>(const char *path, int (*fn) (const char *, const struct stat     *, int), int depth);  int <b>nftw</b>(const char *path, int (*fn) (const char *, const struct     stat *, int, struct FTW*), int depth, int flags);</pre>																		
DESCRIPTION	<p>The <code>ftw()</code> function recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, <code>ftw()</code> 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 <code>stat</code> structure (see <code>stat(2)</code>) containing information about the object, and an integer. Possible values of the integer, defined in the <code>&lt;ftw.h&gt;</code> header, are:</p> <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>The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.</td></tr> </table> <p>The <code>ftw()</code> function visits a directory before visiting any of its descendants.</p> <p>The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a non-zero value, or some error is detected within <code>ftw()</code> (such as an I/O error). If the tree is exhausted, <code>ftw()</code> returns 0. If <i>fn</i> returns a non-zero value, <code>ftw()</code> stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p> <p>The <code>nftw()</code> function 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:</p> <table> <tr> <td>FTW_PHYS</td><td>Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> privileges, the walk visits all SLDs in</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	The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.	FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> privileges, the walk visits all SLDs in
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	The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.																		
FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> 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.

The `nftw()` function calls *fn* with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the `stat` structure (see `stat(2)`) containing information about the object, the third is an integer giving additional information, and the fourth is a `struct FTW` that contains the following members:

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

The `base` member is the offset into the pathname of the base name of the object. The `level` member 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:

<code>FTW_F</code>	The object is a file.
<code>FTW_D</code>	The object is a directory.
<code>FTW_DP</code>	The object is a directory and subdirectories have been visited.
<code>FTW_SL</code>	The object is a symbolic link.
<code>FTW_SLN</code>	The object is a symbolic link that points to a non-existent file.
<code>FTW_DNR</code>	The object is a directory that cannot be read. ]The user-defined function <i>fn</i> will not be called for any of its descendants.
<code>FTW_NS</code>	The <code>stat()</code> function failed on the object because of lack of appropriate permission. The <code>stat</code> buffer passed to <i>fn</i> is undefined. The <code>stat()</code> function failed for a reason other than lack of appropriate permission. <code>EACCES</code> 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. It must not be greater than the number of file descriptors currently available for use. The `ftw()` function 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 the tree is exhausted, `ftw()` and `nftw()` return 0. If the function pointed to by *fn* returns a non-zero value, `ftw()` and `nftw()` stop their tree traversal and return whatever value was returned by the function pointed to by *fn*. If `ftw()` and `nftw()` detect an error, they return `-1` and set `errno` to indicate the error.

If `ftw()` and `nftw()` encounter an error other than `EACCES` (see `FTW_DNR` and `FTW_NS` above), they return `-1` and set `errno` to indicate the error. The external variable `errno` may contain any error value that is possible when a directory is opened or when one of the `stat` functions is executed on a directory or file.

## ERRORS

The `ftw()` and `nftw()` functions will fail if:

<code>ENAMETOOLONG</code>	The length of the path exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> .
<code>ENOENT</code>	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
<code>ENOTDIR</code>	A component of <i>path</i> is not a directory.

The `ftw()` function will fail if:

<code>EACCES</code>	Search permission is denied for any component of <i>path</i> or read permission is denied for <i>path</i> .
<code>ELOOP</code>	Too many symbolic links were encountered.

The `nftw()` function will fail if:

<code>EACCES</code>	Search permission is denied for any component of <i>path</i> or read permission is denied for <i>path</i> , or <i>fn()</i> returns <code>-1</code> and does not reset <code>errno</code> .
---------------------	--

The `ftw()` and `nftw()` functions may fail if:

<code>ENAMETOOLONG</code>	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
---------------------------	---

ftw(3C)

The `ftw()` function may fail if:

`EINVAL` The value of the `ndirs` argument is invalid.

The `nftw()` function may fail if:

`ELOOP` Too many symbolic links were encountered in resolving *path*.

`EMFILE` There are `OPEN_MAX` file descriptors currently open in the calling process.

`ENFILE` Too many files are currently open in the system.

In addition, if the function pointed to by *fn* encounters system errors, `errno` may be set accordingly.

#### USAGE

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

The `ftw()` function 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 non-zero value at its next invocation.

The `ftw()` and `nftw()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

The `ftw()` function is safe in multithreaded applications. The `nftw()` function is safe in multithreaded 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`.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe with exceptions.

#### 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.

#### Trusted Solaris 8 4/01 Reference Manual

`stat(2)`



longjmp(3C), malloc(3C), attributes(5), lf64(5)

ftw(3C)

## getacdir(3BSM)

<b>NAME</b>	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int getacdir( char *dir, int len); int getacmin( int *min_val); int getacflg( char *auditstring, int len); int getacna( char *auditstring, int len); void setac( void); void endac( void);</pre>						
<b>DESCRIPTION</b>	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>						
<b>FILES</b>	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .						
<b>RETURN VALUES</b>	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table> <tr> <td>0</td><td>on success.</td></tr> <tr> <td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr> </table> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <table> <tr> <td>1</td><td>On EOF.</td></tr> </table> <p><code>getacdir()</code> returns:</p>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.	1	On EOF.
0	on success.						
-2	On failure and set <code>errno</code> to indicate the error.						
1	On EOF.						

getacdir(3BSM)

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir().

These functions return:

- 3 If the directory entry format in the audit\_control file is incorrect.
- getacdir(), getacflg() and getacna() return:
- 3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_warn(1M), inetd(1M), audit\_control(4)

attributes(5)

## getacflg(3BSM)

<b>NAME</b>	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int getacdir( char *dir, int len); int getacmin( int *min_val); int getacflg( char *auditstring, int len); int getacna( char *auditstring, int len); void setac( void); void endac( void);</pre>						
<b>DESCRIPTION</b>	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>						
<b>FILES</b>	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .						
<b>RETURN VALUES</b>	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table> <tr> <td>0</td><td>on success.</td></tr> <tr> <td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr> </table> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <table> <tr> <td>1</td><td>On EOF.</td></tr> </table> <p><code>getacdir()</code> returns:</p>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.	1	On EOF.
0	on success.						
-2	On failure and set <code>errno</code> to indicate the error.						
1	On EOF.						

- 1 on EOF.
  - 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.
- These functions return:
- 3 If the directory entry format in the `audit_control` file is incorrect.
- `getacdir()`, `getacflg()` and `getacna()` return:
- 3 If the input buffer is too short to accommodate the record.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

## getacinfo(3BSM)

<b>NAME</b>	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int getacdir( char *dir, int len); int getacmin( int *min_val); int getacflg( char *auditstring, int len); int getacna( char *auditstring, int len); void setac( void); void endac( void);</pre>						
<b>DESCRIPTION</b>	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>						
<b>FILES</b>	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .						
<b>RETURN VALUES</b>	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table> <tr> <td>0</td><td>on success.</td></tr> <tr> <td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr> </table> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <table> <tr> <td>1</td><td>On EOF.</td></tr> </table> <p><code>getacdir()</code> returns:</p>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.	1	On EOF.
0	on success.						
-2	On failure and set <code>errno</code> to indicate the error.						
1	On EOF.						

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 If the directory entry format in the `audit_control` file is incorrect.
- `getacdir()`, `getacflg()` and `getacna()` return:
- 3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

## getacmin(3BSM)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information						
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>ns</code> -l<code>intl</code> [<i>library...</i>]  #include &lt;bsm/libbsm.h&gt;  int <b>getacdir</b>( char *<i>dir</i>, int <i>len</i> ); int <b>getacmin</b>( int *<i>min_val</i> ); int <b>getacflg</b>( char *<i>auditstring</i>, int <i>len</i> ); int <b>getacna</b>( char *<i>auditstring</i>, int <i>len</i> ); void <b>setac</b>( void ); void <b>endac</b>( void );</pre>						
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> 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 <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>						
FILES	<table><tr><td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr></table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .						
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table><tr><td>0</td><td>on success.</td></tr><tr><td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr></table> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <table><tr><td>1</td><td>On EOF.</td></tr></table> <p><code>getacdir()</code> returns:</p>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.	1	On EOF.
0	on success.						
-2	On failure and set <code>errno</code> to indicate the error.						
1	On EOF.						



getacmin(3BSM)

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 If the directory entry format in the `audit_control` file is incorrect.
- `getacdir()`, `getacflg()` and `getacna()` return:
- 3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

## getacna(3BSM)

<b>NAME</b>	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int getacdir( char *dir, int len ); int getacmin( int *min_val ); int getacflg( char *auditstring, int len ); int getacna( char *auditstring, int len ); void setac( void ); void endac( void );</pre>						
<b>DESCRIPTION</b>	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>						
<b>FILES</b>	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .						
<b>RETURN VALUES</b>	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table> <tr> <td>0</td><td>on success.</td></tr> <tr> <td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr> </table> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <table> <tr> <td>1</td><td>On EOF.</td></tr> </table> <p><code>getacdir()</code> returns:</p>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.	1	On EOF.
0	on success.						
-2	On failure and set <code>errno</code> to indicate the error.						
1	On EOF.						

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 If the directory entry format in the `audit_control` file is incorrect.
- `getacdir()`, `getacflg()` and `getacna()` return:
- 3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

## getauclassent(3BSM)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  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>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre>char      *ac_name; au_class_t  ac_class; char      *ac_desc;</pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

getauclassent(3BSM)

getauclassent() and getauclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES** /etc/security/audit\_class Maps audit class numbers to audit class names.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
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.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

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.

## getauclassent\_r(3BSM)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  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>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre>char      *ac_name; au_class_t ac_class; char      *ac_desc;</pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

getauclassent\_r(3BSM)

getauclassent() and getauclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES** /etc/security/audit\_class Maps audit class numbers to audit class names.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
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.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

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.

## getauclassnam(3BSM)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_class_ent *<b>getauclassnam</b>( const char *<i>name</i> );  struct au_class_ent *<b>getauclassnam_r</b>( au_class_ent_t *<i>class_int</i>,     const char *<i>name</i> );  struct au_class_ent *<b>getauclassent</b>( void );  struct au_class_ent *<b>getauclassent_r</b>( au_class_ent_t *<i>class_int</i> );  void <b>setauclass</b>( void );  void <b>endauclass</b>( void ); </pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre> char          *ac_name; au_class_t    ac_class; char          *ac_desc; </pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.



getauclassnam(3BSM)

getauclassent() and getauclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES** /etc/security/audit\_class Maps audit class numbers to audit class names.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
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.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

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.

## getauclassnam\_r(3BSM)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  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>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre>char      *ac_name; au_class_t ac_class; char      *ac_desc;</pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

getaclassnam\_r(3BSM)

getaclassent() and getaclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES** /etc/security/audit\_class Maps audit class numbers to audit class names.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except getaclassent() and getaclassnam(). The two functions, getaclassent\_r() and getaclassnam\_r() have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

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.

## getauditflags(3BSM)

<b>NAME</b>	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  int <b>getauditflagsbin</b>(char *auditstring, au_mask_t *masks);  int <b>getauditflagschar</b>(char *auditstring, au_mask_t *masks, int     verbose);</pre>				
<b>DESCRIPTION</b>	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in audit_control(4).</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in &lt;bsm/audit.h&gt;, which is read by &lt;bsm/libbsm.h&gt;.</p>				
<b>RETURN VALUES</b>	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0            On success.</p> <p>-1           On failure.</p>				
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe.</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b> Trusted Solaris 8 4/01 Reference Manual <b>BUGS</b>	<p>The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.</p> <p>audit.log(4), audit_control(4)</p> <p>attributes(5)</p> <p>This is not a very extensible interface.</p>				

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  int <b>getauditflagsbin</b>(char *auditstring, au_mask_t *masks);  int <b>getauditflagschar</b>(char *auditstring, au_mask_t *masks, int     verbose);</pre>				
DESCRIPTION	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in audit_control(4).</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in &lt;bsm/audit.h&gt;, which is read by &lt;bsm/libbsm.h&gt;.</p>				
RETURN VALUES	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0            On success.</p> <p>-1           On failure.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe.</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual BUGS	<p>The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.</p> <p>audit.log(4), audit_control(4)</p> <p>attributes(5)</p> <p>This is not a very extensible interface.</p>				

## getauditflagschar(3BSM)

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  int getauditflagsbin(char *auditstring, au_mask_t *masks);  int getauditflagschar(char *auditstring, au_mask_t *masks, int     verbose);</pre>				
DESCRIPTION	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in audit_control(4).</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in &lt;bsm/audit.h&gt;, which is read by &lt;bsm/libbsm.h&gt;.</p>				
RETURN VALUES	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0            On success.</p> <p>-1           On failure.</p>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe.</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SUMMARY OF TRUSTED SOLARIS CHANGES SunOS 5.8 Reference Manual BUGS	<p>The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.</p> <p>audit.log(4), audit_control(4)</p> <p>attributes(5)</p> <p>This is not a very extensible interface.</p>				

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endauevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>

getauevent(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

## RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

## FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`



getauevent(3BSM)

**NOTES** All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

## getauevent\_r(3BSM)

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>

getauevent\_r(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a struct `au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauevnam_r(3BSM)`, `audit_class(4)`, `audit_event(4)`  
`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevent\_r(3BSM)

<b>NOTES</b>	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.
--------------	---

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endaeuevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -l<code>bsm</code> -l<code>socket</code> -l<code>ns</code> -l<code>intl</code> [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endaeuevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>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.</p>

getauevnam(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

## RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

## FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevnam(3BSM)

**NOTES** All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

## getauevnam\_r(3BSM)

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>



getauevnam\_r(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauevnam_r(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevnam\_r(3BSM)

<b>NOTES</b>	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.
--------------	---

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endaeuevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endaeuevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>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.</p>

getauevnonam(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

## RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

## FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevnonam(3BSM)

**NOTES** All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

## getauevnum(3BSM)

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endauevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>

getauevnum(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauevnam_r(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevnum(3BSM)

<b>NOTES</b>	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.
--------------	---



NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endaeuevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endaeuevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>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.</p>

getauevnum\_r(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

## RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

## FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

getauevnum\_r(3BSM)

**NOTES** All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

## getauthattr(3SECDB)

NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void); authattr_t *getauthnam(const char *name); void free_authattr(authattr_t *auth); void setauthattr(void); void endauthattr(void); int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The <code>getauthattr()</code> and <code>getauthnam()</code> functions each return an <code>auth_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getauthattr()</code> function enumerates <code>auth_attr</code> entries. The <code>getauthnam()</code> function searches for an <code>auth_attr</code> entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive <code>auth_attr</code> entries or NULL.</p> <p>The internal representation of an <code>auth_attr</code> entry is an <code>authattr_t</code> structure defined in <code>&lt;auth_attr.h&gt;</code> with the following members:</p> <pre>char  *name;           /* name of the authorization */ char  *res1;           /* reserved for future use */ char  *res2;           /* reserved for future use */ char  *short_desc;     /* short description */ char  *long_desc;      /* long description */ kva_t *attr;           /* array of key-value pair attributes */</pre> <p>The <code>setauthattr()</code> function “rewinds” to the beginning of the enumeration of <code>auth_attr</code> entries. Calls to <code>getauthnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setauthattr()</code> should be called before the first call to <code>getauthattr()</code>.</p> <p>The <code>endauthattr()</code> function may be called to indicate that <code>auth_attr</code> processing is complete; the system may then close any open <code>auth_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>chkauthattr()</code> function verifies whether or not a user has a given authorization. It first reads the <code>AUTHS_GRANTED</code> key in the <code>/etc/security/policy.conf</code> file and returns 1 if it finds a match for the given authorization. If <code>chkauthattr()</code> does not find a match, it reads the <code>PROFS_GRANTED</code> key in <code>/etc/security/policy.conf</code> and returns 1 if the given authorization is in any profiles specified with the <code>PROFS_GRANTED</code> keyword. If a</p>

match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	<code>/etc/security/policy.conf</code> or	Is user
Authorization name	<code>user_attr</code> or <code>prof_attr</code> entry	authorized?
<code>solaris.printer.postscript</code>	<code>solaris.printer.postscript</code>	Yes
<code>solaris.printer.postscript</code>	<code>solaris.printer.*</code>	Yes
<code>solaris.printer.grant</code>	<code>solaris.printer.*</code>	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

## RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

## USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

getauthattr(3SECDB)

Individual attributes in the `attr` structure can be referred to by calling the `kva_match(3SECDB)` function.

#### WARNINGS

Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

#### FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/auth_attr</code>	authorization attributes
<code>/etc/security/policy.conf</code>	policy definitions
<code>/etc/security/prof_attr</code>	profile information

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The Trusted Solaris environment adds authorizations. The `chkauthattr()` function replaces the Trusted Solaris 7 `chkauth()` function.

`nsswitch.conf(4)`, `prof_attr(4)`, `user_attr(4)`

`getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `getuserattr(3SECDB)`,  
`kva_match(3SECDB)`, `auth_attr(4)`, `attributes(5)`, `rbac(5)`

NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void);  authattr_t *getauthnam(const char *name);  void free_authattr(authattr_t *auth);  void setauthattr(void);  void endauthattr(void);  int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The <code>getauthattr()</code> and <code>getauthnam()</code> functions each return an <code>auth_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getauthattr()</code> function enumerates <code>auth_attr</code> entries. The <code>getauthnam()</code> function searches for an <code>auth_attr</code> entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive <code>auth_attr</code> entries or NULL.</p> <p>The internal representation of an <code>auth_attr</code> entry is an <code>authattr_t</code> structure defined in <code>&lt;auth_attr.h&gt;</code> with the following members:</p> <pre>char  *name;           /* name of the authorization */ char  *res1;           /* reserved for future use */ char  *res2;           /* reserved for future use */ char  *short_desc;     /* short description */ char  *long_desc;      /* long description */ kva_t *attr;           /* array of key-value pair attributes */</pre> <p>The <code>setauthattr()</code> function “rewinds” to the beginning of the enumeration of <code>auth_attr</code> entries. Calls to <code>getauthnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setauthattr()</code> should be called before the first call to <code>getauthattr()</code>.</p> <p>The <code>endauthattr()</code> function may be called to indicate that <code>auth_attr</code> processing is complete; the system may then close any open <code>auth_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>chkauthattr()</code> function verifies whether or not a user has a given authorization. It first reads the <code>AUTHS_GRANTED</code> key in the <code>/etc/security/policy.conf</code> file and returns 1 if it finds a match for the given authorization. If <code>chkauthattr()</code> does not find a match, it reads the <code>PROFS_GRANTED</code> key in <code>/etc/security/policy.conf</code> and returns 1 if the given authorization is in any profiles specified with the <code>PROFS_GRANTED</code> keyword. If a</p>

getauthnam(3SECDB)

match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	/etc/security/policy.conf <b>or</b>	<b>Is user</b>
Authorization name	<code>user_attr</code> <b>or</b> <code>prof_attr</code> <b>entry</b>	<b>authorized?</b>
solaris.printer.postscript	solaris.printer.postscript	Yes
solaris.printer.postscript	solaris.printer.*	Yes
solaris.printer.grant	solaris.printer.*	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

#### RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

#### USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.



		getauthnam(3SECDB)				
		Individual attributes in the attr structure can be referred to by calling the kva_match(3SECDB) function.				
WARNINGS		Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.				
FILES	/etc/nsswitch.conf	configuration file lookup information for the name server switch				
	/etc/user_attr	extended user attributes				
	/etc/security/auth_attr	authorization attributes				
	/etc/security/policy.conf	policy definitions				
	/etc/security/prof_attr	profile information				
ATTRIBUTES		See attributes(5) for descriptions of the following attributes:				
		<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE					
MT-Level	MT-Safe					
SUMMARY OF TRUSTED SOLARIS CHANGES		The Trusted Solaris environment adds authorizations. The chkauthattr() function replaces the Trusted Solaris 7 chkauth() function.				
Trusted Solaris 8	nsswitch.conf(4), prof_attr(4), user_attr(4)					
4/01 Reference Manual	getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB), auth_attr(4), attributes(5), rbac(5)					
SunOS 5.9						
Reference Manual						

get\_auth\_text(3TSOL)

<b>NAME</b>	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre> cc [flag...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt;  #include &lt;bsm/libbsm.h&gt;  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_t *u, const char     *name); struct au_user_ent *getauuserent_r(au_user_ent_t *u); </pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry. Entries can come from any of the sources specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code>&lt;bsm/libbsm.h&gt;</code> with the following members:</p> <pre> char      *au_name; au_mask_t au_always; au_mask_t au_never; </pre>

getauuserent(3BSM)

## RETURN VALUES

The `getauusernam()` function returns a pointer to a `struct_au_user_ent` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `getauuserent()` function returns a pointer to a `struct_au_user_ent` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

## FILES

`/etc/security/audit_user` Stores per-user audit event mask.  
`/etc/passwd` Stores user-id to username mappings.

## SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8  
4/01 Reference  
Manual  
Solaris 9  
Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_user(4)`, `nsswitch.conf(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

## NOTES

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

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre> <b>cc</b> [<i>flag...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt;  #include &lt;bsm/libbsm.h&gt;  struct au_user_ent *<b>getauusernam</b>(const char *<i>name</i>); struct au_user_ent *<b>getauuserent</b>(void); void <b>setauuser</b>(void); void <b>endauuser</b>(void);  struct au_user_ent *<b>getauusernam_r</b>(au_user_ent_t *<i>u</i>, const char     *<i>name</i>); struct au_user_ent *<b>getauuserent_r</b>(au_user_ent_t *<i>u</i>); </pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry. Entries can come from any of the sources specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or NULL.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code>&lt;bsm/libbsm.h&gt;</code> with the following members:</p> <pre> char      *au_name; au_mask_t au_always; au_mask_t au_never; </pre>

getauusernam(3BSM)

**RETURN VALUES**

The `getauusernam()` function returns a pointer to a `struct_au_user_ent` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `getauuserent()` function returns a pointer to a `struct_au_user_ent` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

**FILES**

<code>/etc/security/audit_user</code>	Stores per-user audit event mask.
<code>/etc/passwd</code>	Stores user-id to username mappings.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

Trusted Solaris 8  
4/01 Reference  
Manual  
Solaris 9  
Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_user(4)`, `nsswitch.conf(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

**NOTES**

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

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

<b>NAME</b>	blportion, bcltos1, getcs1, setcs1 – access binary label portions						
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  bslabel_t *bcltos1(bslabel_t *label);  void getcs1(bslabel_t *destination_label, const bslabel_t *source_label); void setcs1(bslabel_t *destination_label, const bslabel_t *source_label);</pre>						
<b>DESCRIPTION</b>	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltos1() provides a pointer to the sensitivity label of the binary CMW label <i>label</i>.</p> <p>getcs1() copies the sensitivity label of the binary CMW label <i>source_label</i> to the binary sensitivity label <i>destination_label</i>.</p> <p>setcs1() replaces the value of the sensitivity label of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label <i>source_label</i>.</p>						
<b>RETURN VALUES</b>	bcltos1() returns a pointer to its label type.						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> Comparing Sensitivity Labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blequal(bcltos1(&amp;cmw_label), &amp;file_sensitivity_label)</pre>						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<pre>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</pre> <p><i>Trusted Solaris Developer's Guide</i></p>						
<b>SunOS 5.8 Reference Manual</b>	attributes(5)						

## getexecattr(3SECDB)

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or <code>NULL</code>. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to <code>NULL</code>.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                 /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>



The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See **EXAMPLES**.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

## RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and

getexecattr(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

## EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL)) == NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL)) == NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (`*`), if defined, is returned.

```
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE)) == NULL) {
    /* do error */
}
```

## FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

getexecattr(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

## getexecprof(3SECDB)

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdB -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or <code>NULL</code>. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to <code>NULL</code>.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                   /* relevant */ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>

The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See **EXAMPLES**.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

## RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and

## getexecprof(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

### EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL)) == NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL)) == NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (`*`), if defined, is returned.

```
if ((execprof=getexecprof("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE)) == NULL) {
    /* do error */
}
```

### FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

getexecprof(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

## getexecuser(3SECDB)

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or <code>NULL</code>. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to <code>NULL</code>.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                 /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>



The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See **EXAMPLES**.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

## RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and

## getexecuser(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

### EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (\*), if defined, is returned.

```
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

### FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

getexecuser(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

## getfauditflags(3BSM)

<b>NAME</b>	getfauditflags – Generates the process audit state				
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  int <b>getfauditflags</b>(au_mask_t *usremasks, au_mask_t *usrdmasks,     au_mask_t *lastmasks);</pre>				
<b>DESCRIPTION</b>	<p>getfauditflags() generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the audit_control(4) file. getfauditflags() obtains the system audit value by calling getacflg() (see getacinfo(3BSM).)</p> <p>usremasks points to au_mask_t fields which contains 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 always to be audited when they fail.</p> <p>usrdmasks also points to au_mask_t fields which contains 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 never to be audited when they fail.</p> <p>The structures pointed to by usremasks and usrdmasks may be obtained from the audit_user(4) file by calling getauusernam() which returns a pointer to a structure containing all audit_user(4) fields for a user.</p> <p>The output of this function is stored in lastmasks which is a pointer of type au_mask_t as well. The first value defines which events are to be audited when they succeed and the second defines which events are to be audited when they fail.</p> <p>Both usremasks and usrdmasks override the values in the system audit values.</p>				
<b>RETURN VALUES</b>	–1 is returned on error and 0 on success.				
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe.</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b> Trusted Solaris 8 4/01 Reference Manual  SunOS 5.8 Reference Manual	<p>The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.</p> <p>getacinfo(3BSM), getauditflags(3BSM), getauusernam(3BSM), audit.log(4), audit_control(4), audit_user(4)</p> <p>attributes(5)</p>				

NAME	getpeerinfo – Get peer’s process characteristics
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/audit.h&gt;  int <b>getpeerinfo</b>(int <i>fd</i>, au_peergroupinfo_t *<i>grpinfo</i>,                  au_peermiscinfo_t *<i>peerinfo</i>);</pre>
DESCRIPTION	<p>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.</p> <p>The au_peergroupinfo structure has the following form:</p> <pre>struct peergroupinfo {     ulong_t    peer_ngroups           /* number of elements obtained */     gid_t      peer_groups[NGROUPS_UMAX]; /* peer’s supplemental groups */ };</pre> <p>The remaining attributes are returned in <i>peerinfo</i> which is of type struct au_peermiscinfo and has been allocated by the calling process. The au_peermiscinfo structure has the following form:</p> <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> <p>where auditinfo_t is of type struct auditinfo which has the following form:</p> <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> <p>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 <i>peerinfo</i> attributes from its peer. No privileges are required to obtain just <i>grpinfo</i>.</p>
RETURN VALUES	getpeerinfo() returns 0 on success. On failure it returns a negative value and sets errno to indicate the error.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

getpeerinfo(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl
MT-Level	MT-Safe

#### 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.

#### NOTES

Available only on Trusted Solaris systems with auditing enabled. Auditing is enabled by default in the Trusted Solaris environment.

**SunOS 5.8  
Reference Manual**

attributes(5)

NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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.</p> <p>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 <code>file_mac_read,file_mac_write,none,proc_nofloat</code> produces the same result as <code>proc_nofloat</code> alone. The constructed privilege set is stored in the <i>priv_set_t</i> buffer to which <i>priv_set</i> points.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

get\_priv\_text(3TSOL)

RETURN VALUES	get_priv_text()	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
	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 NULL. The function returns NULL 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 TSOL_MAX_PRIV.
	str_to_priv_set()	Returns NULL 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.
ERRORS	priv_to_str() may fail for this reason:	
	EINVAL	The specified <i>priv_id</i> is greater than TSOL_MAX_PRIV.
	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.
NOTES	EFAULT	The specified <i>priv_name</i> is an invalid address.
	To use these routines, the program must be loaded with the Trusted Solaris library libtsol or libtsol.so.	
Trusted Solaris 8 4/01 Reference Manual	priv_desc(4)	priv_name(4)
	attributes(5)	



NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [ flag... ] file... -lsecdB -lsocket -lnsl -lintl [ library... ] #include &lt;prof.h&gt;  profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	<p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions each return a <code>prof_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getprofattr()</code> function enumerates <code>prof_attr</code> entries. The <code>getprofnam()</code> function searches for a <code>prof_attr</code> entry with a given <i>name</i>. Successive calls to these functions return either successive <code>prof_attr</code> entries or NULL.</p> <p>The internal representation of a <code>prof_attr</code> entry is a <code>profattr_t</code> structure defined in <code>&lt;prof_attr.h&gt;</code> with the following members:</p> <pre>char    name;    /* Name of the profile */ char    res1;    /* Reserved for future use */ char    res2;    /* Reserved for future use */ char    desc;    /* Description/Purpose of the profile */ kva_t   attr;    /* Profile attributes */</pre> <p>The <code>free_profattr()</code> function releases memory allocated by the <code>getprofattr()</code> and <code>getprofnam()</code> functions.</p> <p>The <code>setprofattr()</code> function “rewinds” to the beginning of the enumeration of <code>prof_attr</code> entries. Calls to <code>getprofnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setprofattr()</code> should be called before the first call to <code>getprofattr()</code>.</p> <p>The <code>endprofattr()</code> function may be called to indicate that <code>prof_attr</code> processing is complete; the system may then close any open <code>prof_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>getproflist()</code> function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by <code>getproflist()</code> should be freed using the <code>free_proflist()</code> function.</p>

getprofattr(3SECDB)

	<p>The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.</p>				
RETURN VALUES	<p>The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>				
USAGE	<p>Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function.</p> <p>Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error.</p> <p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function.</p> <p>Applications 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. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.</p>				
FILES	<code>/etc/security/prof_attr</code> profiles and their descriptions				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>auths(1)</code> , <code>profiles(1)</code> , <code>getexecattr(3SECDB)</code> , <code>getauthattr(3SECDB)</code> , <code>prof_attr(4)</code>				

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
DESCRIPTION	The getprofent, setprofent, endprofent, getprofentbyname, and free_profent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

getprofentbyname(3TSOL)

<b>NAME</b>	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get user profile description
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getprofent, setprofent, endprofent, getprofentbyname, and free_profent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [ flag... ] file... -lsecdB -lsocket -lnsl -lintl [ library... ] #include &lt;prof.h&gt;  profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	<p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions each return a <code>prof_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getprofattr()</code> function enumerates <code>prof_attr</code> entries. The <code>getprofnam()</code> function searches for a <code>prof_attr</code> entry with a given <i>name</i>. Successive calls to these functions return either successive <code>prof_attr</code> entries or NULL.</p> <p>The internal representation of a <code>prof_attr</code> entry is a <code>profattr_t</code> structure defined in <code>&lt;prof_attr.h&gt;</code> with the following members:</p> <pre>char    name;    /* Name of the profile */ char    res1;    /* Reserved for future use */ char    res2;    /* Reserved for future use */ char    desc;    /* Description/Purpose of the profile */ kva_t   attr;    /* Profile attributes */</pre> <p>The <code>free_profattr()</code> function releases memory allocated by the <code>getprofattr()</code> and <code>getprofnam()</code> functions.</p> <p>The <code>setprofattr()</code> function “rewinds” to the beginning of the enumeration of <code>prof_attr</code> entries. Calls to <code>getprofnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setprofattr()</code> should be called before the first call to <code>getprofattr()</code>.</p> <p>The <code>endprofattr()</code> function may be called to indicate that <code>prof_attr</code> processing is complete; the system may then close any open <code>prof_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>getproflist()</code> function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by <code>getproflist()</code> should be freed using the <code>free_proflist()</code> function.</p>

## getprofnam(3SECDB)

	<p>The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.</p>				
RETURN VALUES	<p>The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>				
USAGE	<p>Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function.</p> <p>Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error.</p> <p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function.</p> <p>Applications 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. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.</p>				
FILES	<code>/etc/security/prof_attr</code> profiles and their descriptions				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>auths(1)</code> , <code>profiles(1)</code> , <code>getexecattr(3SECDB)</code> , <code>getauthattr(3SECDB)</code> , <code>prof_attr(4)</code>				

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] ( <b>obsolete</b> )
DESCRIPTION	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

## getprofstrbyname(3TSOL)

<b>NAME</b>	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).



NAME	getsockopt, setsockopt – get and set options on sockets
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int <b>getsockopt</b>(int s, int level, int optname, void *optval, int *optlen); int <b>setsockopt</b>(int s, int level, int optname, const void *optval, int                optlen);</pre>
DESCRIPTION	<p>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.</p> <p>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(3SOCKET)).</p> <p>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.</p> <p><i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file &lt;sys/socket.h&gt; contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</p> <p>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 &lt;sys/socket.h&gt;. struct linger contains the following members:</p> <pre>l_onoff          on = 1/off = 0 l_linger         linger time, in seconds</pre> <p>The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt().</p> <pre>SO_DEBUG         enable/disable recording of debugging information SO_REUSEADDR     enable/disable local address reuse SO_KEEPALIVE     enable/disable keep connections alive</pre>

## getsockopt(3SOCKET)

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_DGRAM_ERRIND	application wants delayed error
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(3SOCKET)` 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. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

getsockopt(3SOCKET)

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. `SO_DGRAM_ERRIND` makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (*to* parameter) to fail with the appropriate error. See `send(3SOCKET)`.

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.

## RETURN VALUES

`getsockopt()` returns:

0            On success.

-1           On failure, and sets `errno` to indicate the error.

## ERRORS

The call succeeds unless:

`EBADF`                      The argument *s* 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.

`ENOSR`                      There were insufficient STREAMS resources available for the operation to complete.

`ENOTSOCK`                   The argument *s* is not a socket.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

## 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
Sum 1998  
Reference Manual

`read(2)`, `bind(3SOCKET)`, `send(3SOCKET)`, `socket(3SOCKET)`

`close(2)`, `ioctl(2)`, `getprotobyname(3SOCKET)`, `recv(3SOCKET)`, `netconfig(4)`, `attributes(5)`

## getuserattr(3SECDB)

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> - lsecdb - lsocket - lns1 - lint1 [ <i>library...</i> ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>

**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/user_attr</code>	extended user attributes
<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

getuserent(3TSOL)

<b>NAME</b>	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ]  ( <b>obsolete</b> )
<b>DESCRIPTION</b>	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).

getuserentbyname(3TSOL)

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ]  ( <b>obsolete</b> )
DESCRIPTION	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).

## getuserentbyuid(3TSOL)

<b>NAME</b>	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).



NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ flag... ] file... - lsecdb - lsocket - lns1 - lint1 [ library... ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>

## getusernam(3SECDB)

**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES** `/etc/user_attr` extended user attributes  
`/etc/nsswitch.conf` configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ flag... ] file... - lsecdb - lsocket - lns1 - lint1 [ library... ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>

getuserid(3SECDB)

**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/user_attr</code>	extended user attributes
<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];      /* user login name */ char          ut_id[4];       /* /sbin/inittab id (usually line #) */ char          ut_line[12];    /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;           /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

## getutent(3C)

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

<p>SUMMARY OF TRUSTED SOLARIS CHANGES</p> <p>SunOS 5.8 Reference Manual</p> <p>NOTES</p>	<p style="text-align: right;">getutent(3C)</p> <p>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.</p> <p>ttyslot(3C), utmp(4), utmpx(4), attributes(5)</p> <p>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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().</p>
--	--

## getutid(3C)

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];    /* user login name */ char          ut_id[4];      /* /sbin/inittab id (usually line #) */ char          ut_line[12];   /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;            /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>



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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

getutid(3C)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

**SunOS 5.8  
Reference Manual  
NOTES**

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.

ttyslot(3C), utmp(4), utmpx(4), attributes(5)

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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];    /* user login name */ char          ut_id[4];     /* /sbin/inittab id (usually line #) */ char          ut_line[12];  /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;           /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

## getutline(3C)

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

**SunOS 5.8  
Reference Manual**

**NOTES**

getutline(3C)

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.

ttyslot(3C), utmp(4), utmpx(4), attributes(5)

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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().

## getutmp(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## getutmp(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the <i>e_termination</i> and <i>e_exit</i> members of the <i>ut_exit</i> structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by <i>init(1M)</i>, these values are set according to the result of the <i>wait()</i> call that <i>init</i> performs on the process when the process exits. See the <i>wait(2)</i> manual page for the values <i>init</i> uses. Applications creating utmpx entries can set <i>ut_exit</i> values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See <i>wstat(3XFN)</i> for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The <i>ut_session</i> member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the <i>nonuser()</i> and <i>nonuserx()</i> macros use the value of the <i>ut_exit.e_exit</i> member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each <i>pty</i> to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that <i>login</i> places in the <i>ut_exit.e_exit</i> member.</p>
RETURN VALUES	<p>Upon successful completion, <i>getutxent()</i>, <i>getutxid()</i>, and <i>getutxline()</i> each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to <i>getutxid()</i> or <i>getutxline()</i>.</p> <p>Upon successful completion, <i>pututxline()</i> returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>



The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

Trusted Solaris 8  
4/01 Reference  
Manual  
NOTES

`getutent(3C)`

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## getutmpx(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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 utmp *utmp, struct utmpx *utmpx);  void getutmpx(struct utmp *utmp, struct utmpx *utmpx);  void updwtmp(char *wfile, struct utmp *utmp);  void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## getutmpx(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference**

**Manual**

**Reference Manual**

**NOTES**

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## getutxent(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## getutxent(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the <code>e_termination</code> and <code>e_exit</code> members of the <code>ut_exit</code> structure are valid only for records of type <code>DEAD_PROCESS</code>. For utmpx entries created by <code>init(1M)</code>, these values are set according to the result of the <code>wait()</code> call that <code>init</code> performs on the process when the process exits. See the <code>wait(2)</code> manual page for the values <code>init</code> uses. Applications creating utmpx entries can set <code>ut_exit</code> values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See <code>wstat(3XFN)</code> for descriptions of the <code>WTERMSIG</code> and <code>WEXITSTATUS</code> macros.</p> <p>The <code>ut_session</code> member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type <code>USER_PROCESS</code>, the <code>nonuser()</code> and <code>nonuserx()</code> macros use the value of the <code>ut_exit.e_exit</code> member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each <code>pty</code> to have a utmpx record (as most applications expect.). The <code>NONROOT_USER</code> macro defines the value that <code>login</code> places in the <code>ut_exit.e_exit</code> member.</p>
RETURN VALUES	<p>Upon successful completion, <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to <code>getutxid()</code> or <code>getutxline()</code>.</p> <p>Upon successful completion, <code>pututxline()</code> returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>



The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8  
4/01 Reference  
Manual  
NOTES

`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

`getutent(3C)`

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## getutxid(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## getutxid(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference**

**Manual**

**Reference Manual**

**NOTES**

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## getutxline(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## getutxline(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>



The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

Trusted Solaris 8  
4/01 Reference  
Manual  
NOTES

`getutent(3C)`

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

getvfsaent(3TSOL)

NAME	getvfsaent, getvfsafile – Get vfstab_adjunct file entry						
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;stdio.h&gt; #include &lt;tsol/vfstab_adjunct.h&gt;  int getvfsaent(FILE *fp, struct vfsaent **vp); int getvfsafile(FILE *fp, struct vfsaent **vp, char *file);</pre>						
DESCRIPTION	<p>getvfsaent () and getvfsafile () 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>&lt;tsol/vfstab_adjunct.h&gt;</code> header:</p> <pre>struct vfsaent {     char    *vfsa_fsname;     char    *vfsa_attrs; };</pre> <p>The <i>vfsa_fsname</i> contains the full pathname of the file system as listed in vfstab(4). The <i>vfsa_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(4).</p> <p>getvfsaent () returns a pointer to the next vfsaent structure in the file; so successive calls can be used to search the entire file. getvfsafile () searches the file referred to by <i>fp</i> until a mount point matching <i>file</i> is found.</p> <p>On successful return, the locations referred to by <i>*vp</i>, <i>*vp-&gt;vfsa_fsname</i>, and <i>*vp-&gt;vfsa_attrs</i> have been separately allocated and may be independently released by calls to free(3C).</p> <p>Note that these routines do not open, close, or rewind the file.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>If the next entry is successfully read by getvfsaent () or a match is found with getvfsafile (), 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:</p> <table><tbody><tr><td>ENOMEM</td><td>Memory cannot be allocated for an entry.</td></tr></tbody></table>	ENOMEM	Memory cannot be allocated for an entry.				
ENOMEM	Memory cannot be allocated for an entry.						

		getvfaent(3TSOL)
	<b>FILES</b>	/etc/security/tsol/vfstab_adjunct Attribute data file for mounting a file system in the Trusted Solaris environment.
<b>Trusted Solaris 8 4/01 Reference Manual</b>		vfstab_adjunct(4)
<b>Trusted Solaris 8 4/01 Reference Manual</b>		>attributes(5)
	<b>NOTES</b>	These interfaces are uncommitted, which means that even though they are not expected to change, they may change between minor releases of the Trusted Solaris environment.

getvfsafile(3TSOL)

NAME	getvfsaent, getvfsafile – Get vfstab_adjunct file entry						
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;stdio.h&gt; #include &lt;tsol/vfstab_adjunct.h&gt;  int getvfsaent(FILE *fp, struct vfsaent **vp); int getvfsafile(FILE *fp, struct vfsaent **vp, char *file);</pre>						
DESCRIPTION	<p>getvfsaent () and getvfsafile () 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>&lt;tsol/vfstab_adjunct.h&gt;</code> header:</p> <pre>struct vfsaent {     char    *vfsa_fsnname;     char    *vfsa_attrs; };</pre> <p>The <i>vfsa_fsnname</i> contains the full pathname of the file system as listed in vfstab(4). The <i>vfsa_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(4).</p> <p>getvfsaent () returns a pointer to the next vfsaent structure in the file; so successive calls can be used to search the entire file. getvfsafile () searches the file referred to by <i>fp</i> until a mount point matching <i>file</i> is found.</p> <p>On successful return, the locations referred to by <i>*vp</i>, <i>*vp~&gt;vfsa_fsnname</i>, and <i>*vp~&gt;vfsa_attrs</i> have been separately allocated and may be independently released by calls to free(3C).</p> <p>Note that these routines do not open, close, or rewind the file.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>If the next entry is successfully read by getvfsaent () or a match is found with getvfsafile (), 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:</p> <table><tbody><tr><td>ENOMEM</td><td>Memory cannot be allocated for an entry.</td></tr></tbody></table>	ENOMEM	Memory cannot be allocated for an entry.				
ENOMEM	Memory cannot be allocated for an entry.						

		getvfsafire(3TSOL)
	<b>FILES</b>	/etc/security/tsol/vfstab_adjunct Attribute data file for mounting a file system in the Trusted Solaris environment.
<b>Trusted Solaris 8 4/01 Reference Manual</b>		vfstab_adjunct(4) >attributes(5)
<b>NOTES</b>		These interfaces are uncommitted, which means that even though they are not expected to change, they may change between minor releases of the Trusted Solaris environment.

## grantpt(3C)

NAME	grantpt – grant access to the slave pseudo-terminal device						
SYNOPSIS	<pre>#include &lt;stdlib.h&gt;  int <b>grantpt</b>(int <i>fildev</i>);</pre>						
DESCRIPTION	<p>The <code>grantpt()</code> function changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counter part. <i>fildev</i> is the file descriptor returned from a successful open of the master pseudo-terminal device. A <i>setuid</i> root program (see <code>setuid(2)</code>) is invoked to change the permissions. The user ID of the slave is set to the real UID of the calling process and the group ID is set to a reserved group. The permission mode of the slave pseudo-terminal is set to readable and writable by the owner and writable by the group.</p> <p>In the Trusted Solaris environment, the <code>grantpt()</code> function has been modified to do nothing and always return successfully. Automatic device allocation obsoletes this function since device ownership and permissions are automatically assigned to the <code>open()</code> process.</p>						
RETURN VALUES	Upon successful completion, <code>grantpt()</code> returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate the error.						
ERRORS	<p>The <code>grantpt()</code> function may fail if:</p> <table><tr><td>EBADF</td><td>The <i>fildev</i> argument is not a valid open file descriptor.</td></tr><tr><td>EINVAL</td><td>The <i>fildev</i> argument is not associated with a master pseudo-terminal device.</td></tr><tr><td>EACCES</td><td>The corresponding slave pseudo-terminal device could not be accessed.</td></tr></table>	EBADF	The <i>fildev</i> argument is not a valid open file descriptor.	EINVAL	The <i>fildev</i> argument is not associated with a master pseudo-terminal device.	EACCES	The corresponding slave pseudo-terminal device could not be accessed.
EBADF	The <i>fildev</i> argument is not a valid open file descriptor.						
EINVAL	The <i>fildev</i> argument is not associated with a master pseudo-terminal device.						
EACCES	The corresponding slave pseudo-terminal device could not be accessed.						
USAGE	The <code>grantpt()</code> function will fail if it is unable to successfully invoke the <i>setuid</i> root program. It may also fail if the application has installed a signal handler to catch SIGCHLD signals.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES 4/01 Reference Manual	<p>Automatic device allocation obsoletes the <code>grantpt()</code> function.</p> <p><code>open(2)</code>, <code>setuid(2)</code></p> <p><code>ptsname(3C)</code>, <code>unlockpt(3C)</code>, <code>attributes(5)</code></p> <p><i>STREAMS Programming Guide</i></p>						

NAME	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <pre>SUN_CMW_ID      label is a binary CMW label. SUN_SL_ID       label is a binary sensitivity label. SUN_CLR_ID      label is a binary clearance.</pre> <p>h_free() frees memory allocated by h_alloc().</p>
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.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

h\_alloc(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

**Trusted Solaris 8  
4/01 Reference  
Manual**

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and bcleartoh\_r() should be used.



NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## herror(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

herror(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

herror(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

herror(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

<b>NAME</b>	hextob, htobcl, htobsl, htobclear – convert hexadecimal string to binary label						
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int htobcl(const char *s, bclabel_t *label) ; int htobsl(const char *s, bslabel_t *label) ; int htobclear(const char *s, bclear_t *clearance) ;</pre>						
<b>DESCRIPTION</b>	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl () converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p><b>Note</b> – The argument to the htobcl () function must contain a fixed <i>n</i>-character hexadecimal number before the sensitivity label hexadecimal value. The fixed number is ignored. CMW labels retain the number for backward compatibility.</p> <p>htobsl () converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>htobclear () converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0xclearance_hexadecimal_value</pre>						
<b>RETURN VALUES</b>	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL), blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL), sbltos(3TSOL)						
	<i>Trusted Solaris Developer's Guide</i>						
<b>SunOS 5.8 Reference Manual</b>	attributes(5)						

h\_free(3TSOL)

NAME	btohex, bcltoh, bsltoh, bcleartoh, bcltoh_r, bsltoh_r, bcleartoh_r, h_alloc, h_free – convert binary label to hexadecimal						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_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 *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>						
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <pre>0xclearance_hexadecimal_value</pre> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <table><tr><td>SUN_CMW_ID</td><td><i>label</i> is a binary CMW label.</td></tr><tr><td>SUN_SL_ID</td><td><i>label</i> is a binary sensitivity label.</td></tr><tr><td>SUN_CLR_ID</td><td><i>label</i> is a binary clearance.</td></tr></table> <p>h_free() frees memory allocated by h_alloc().</p>	SUN_CMW_ID	<i>label</i> is a binary CMW label.	SUN_SL_ID	<i>label</i> is a binary sensitivity label.	SUN_CLR_ID	<i>label</i> is a binary clearance.
SUN_CMW_ID	<i>label</i> is a binary CMW label.						
SUN_SL_ID	<i>label</i> is a binary sensitivity label.						
SUN_CLR_ID	<i>label</i> is a binary clearance.						
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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						



h\_free(3TSOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions

Trusted Solaris 8  
4/01 Reference  
Manual

atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL),  
blmanifest(3TSOL), bltocolor(3TSOL), bltype(3TSOL), labelinfo(3TSOL),  
sbltos(3TSOL)

*Trusted Solaris Developer's Guide*

SunOS 5.8  
Reference Manual  
NOTES

attributes(5)

The functions bcltoh(), bsltoh(), 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(), and bcleartoh\_r() should be used.

## hstrerror(3RESOLV)

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket  -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

```

int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);

```

**DESCRIPTION**

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with RES_USEVC to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

## hstrerror(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>	The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.	
<b>res_nquery, res_query</b>	The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply	

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

## hstrerror(3RESOLV)

<b>res_hostalias</b>	<p>The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the <code>HOSTALIASES</code> environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, <code>NULL</code> is returned. <code>res_hostalias()</code> stores the result in <i>buf</i>.</p>
<b>res_nclose</b>	<p>The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i>.</p>
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or <code>-1</code> if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is <code>NULL</code>, names are not compressed. If <i>lastdnptr</i> is <code>NULL</code>, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	<p>The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i>, which is of size <i>length</i>. <code>dn_expand()</code> returns the size of the compressed name, or <code>-1</code> if there was an error.</p>
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<p><code>/etc/resolv.conf</code> Resolver configuration file</p>
<b>ATTRIBUTES</b>	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p>

hstrerror(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

# SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

## Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

## NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

## htobcl(3TSOL)

<b>NAME</b>	hextob, htobcl, htobsl, htobclear – convert hexadecimal string to binary label						
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int <b>htobcl</b>(const char *s, bclabel_t *label) ; int <b>htobsl</b>(const char *s, bslabel_t *label) ; int <b>htobclear</b>(const char *s, bclear_t *clearance) ;</pre>						
<b>DESCRIPTION</b>	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl () converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p><b>Note</b> – The argument to the htobcl () function must contain a fixed <i>n</i>-character hexadecimal number before the sensitivity label hexadecimal value. The fixed number is ignored. CMW labels retain the number for backward compatibility.</p> <p>htobsl () converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>htobclear () converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0xclearance_hexadecimal_value</pre>						
<b>RETURN VALUES</b>	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL), blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL), sbltos(3TSOL)						
<b>SunOS 5.8 Reference Manual</b>	attributes(5)						
	<i>Trusted Solaris Developer's Guide</i>						



NAME	hextob, htobcl, htobsl, htobclear – convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int htobcl(const char *s, bclabel_t *label) ; int htobsl(const char *s, bslabel_t *label) ; int htobclear(const char *s, bclear_t *clearance) ;</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl () converts into a binary CMW label, a hexadecimal string of the form:</p> <p>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</p> <p><b>Note</b> – The argument to the htobcl () function must contain a fixed <i>n</i>-character hexadecimal number before the sensitivity label hexadecimal value. The fixed number is ignored. CMW labels retain the number for backward compatibility.</p> <p>htobsl () converts into a binary sensitivity label, a hexadecimal string of the form:</p> <p>[0xsensitivity_label_hexadecimal_value]</p> <p>htobclear () converts into a binary clearance, a hexadecimal string of the form:</p> <p>0xclearance_hexadecimal_value</p>						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL), blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL), sbltos(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<p>attributes(5)</p>						

htobsl(3TSOL)

NAME	hextob, htobcl, htobsl, htobclear – convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int htobcl(const char *s, bclabel_t *label);  int htobsl(const char *s, bslabel_t *label);  int htobclear(const char *s, bclear_t *clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0xADMIN_LOW_hex_value [0xsensitivity_label_hexadecimal_value]</pre> <p><b>Note</b> – The argument to the htobcl() function must contain a fixed <i>n</i>-character hexadecimal number before the sensitivity label hexadecimal value. The fixed number is ignored. CMW labels retain the number for backward compatibility.</p> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>[0xsensitivity_label_hexadecimal_value]</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0xclearance_hexadecimal_value</pre>						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3TSOL), blmanifest(3TSOL), bltocolour(3TSOL), bltype(3TSOL), labelinfo(3TSOL), sbltos(3TSOL)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	attributes(5)						

NAME	initgroups – Initialize the supplementary group access list				
SYNOPSIS	<pre>#include &lt;grp.h&gt; #include &lt;sys/types.h&gt;  int <b>initgroups</b>(const char *name, gid_t basegid);</pre>				
DESCRIPTION	<p>The <code>initgroups()</code> function reads the group database to get the group membership for the user specified by <i>name</i>, and initializes the supplementary group access list of the calling process (see <code>getgrnam(3C)</code> and <code>getgroups(2)</code>). 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.</p> <p>While scanning the group database, if the number of groups, including the <i>basegid</i> entry, exceeds <code>NGROUPS_MAX</code>, subsequent group entries are ignored.</p>				
RETURN VALUES	<p><code>initgroups()</code> returns:</p> <p>0            On success.</p> <p>-1           On failure, and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>initgroups()</code> function will fail and not change the supplementary group access list if:</p> <p><code>EPERM</code>            The effective user ID is not superuser.</p>				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>To succeed, <code>initgroups()</code> must have <code>PRIV_PROC_SETID</code> in its set of effective privileges.</p> <p><code>getgroups(2)</code></p> <p><code>getgrnam(3C)</code>, <code>attributes(5)</code></p>				

Trusted Solaris 8  
4/01 Reference Manual  
Solaris 9  
Reference Manual

kstat\_read(3KSTAT)

NAME	kstat_read, kstat_write – Read or write kstat data
SYNOPSIS	<pre>cc [flag...] file ... -lkstat [library...]  #include &lt;kstat.h&gt;  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	<p>kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp-&gt;ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp-&gt;ks_ndata</i> is set to the number of data fields, <i>ksp-&gt;ks_data_size</i> is set to the total size of the data, and <i>ksp-&gt;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-&gt;ks_data</i> into <i>buf</i>.</p> <p>kstat_write() writes data from <i>buf</i>, or from <i>ksp-&gt;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.</p>
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 SunOS 5.8 Reference Manual	<p>kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.</p> <p>kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_close(3KSTAT), kstat_data_lookup(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT)</p>

<b>NAME</b>	kstat_read, kstat_write – Read or write kstat data
<b>SYNOPSIS</b>	<pre>cc [flag...] file ... -lkstat [library...]  #include &lt;kstat.h&gt;  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>
<b>DESCRIPTION</b>	<p>kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp-&gt;ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp-&gt;ks_ndata</i> is set to the number of data fields, <i>ksp-&gt;ks_data_size</i> is set to the total size of the data, and <i>ksp-&gt;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-&gt;ks_data</i> into <i>buf</i>.</p> <p>kstat_write() writes data from <i>buf</i>, or from <i>ksp-&gt;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.</p>
<b>RETURN VALUES</b>	On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return -1.
<b>FILES</b>	/dev/kstat      Kernel statistics driver
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.
<b>Reference Manual</b>	kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_close(3KSTAT), kstat_data_lookup(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT)

kva\_match(3SECDB)

NAME	kva_match – look up a key in a key-value array				
SYNOPSIS	<pre>cc [ flag... ] file... -lsecdb [ library... ] #include &lt;secdb.h&gt;  char *kva_match(kva_t *kva, char *, key);</pre>				
DESCRIPTION	The kva_match() function searches a kva_t structure, which is part of the authattr_t, execattr_t, profattr_t, or userattr_t structures. The function takes two arguments: a pointer to a key value array, and a key. If the key is in the array, the function returns a pointer to the first corresponding value that matches that key. Otherwise, the function returns NULL.				
RETURN VALUES	Upon success, the function returns a pointer to the value sought. Otherwise, it returns NULL.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	getauthattr(3SECDB), getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB)				
NOTES	The kva_match() function returns a pointer to data that already exists in the key-value array. It does not allocate its own memory for this pointer but obtains it from the key-value array that is passed as its first argument.				

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 [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/ModLabel.h&gt;  ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler)()     ok_callback lbuild_attributes extended_operation, ..., NULL);  void *tsol_lbuild_get(ModLabelData *data, lbuild_attributes     extended_operation);  void tsol_lbuild_set(ModLabelData *data lbuild_attributes     extended_operation, ..., NULL);  void tsol_lbuild_destroy(ModLabelData *data);</pre>
DESCRIPTION	<p>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 <code>label_encodings(4)</code> 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.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, the label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>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 grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> 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 <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by the label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The label builder dialog box.</p> <p><code>ok</code> The OK pushbutton.</p> <p><code>cancel</code> The Cancel pushbutton.</p>

## labelbuilder(3TSOL)

reset	The Reset pushbutton.
help	The Help pushbutton.

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

widget	The widget from which the dialog box is created. Any Motif widget can be passed.
ok_callback	A callback function that implements the behavior of the OK pushbutton on the dialog box.
..., NULL	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 `ModLabelData` 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 information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

<code>LBUILD_MODE_IL</code>	Build an information label.
	An information label is fixed at <code>ADMIN_LOW</code> .
<code>LBUILD_MODE_SL</code>	Build a sensitivity label.
<code>LBUILD_MODE_CMW</code>	Build a CMW label.
<code>LBUILD_MODE_CLR</code>	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 buider 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.

## labelbuilder(3TSOL)

	<p><b>LBUILD_LOWER_BOUND</b></p> <p>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.</p> <p><b>LBUILD_UPPER_BOUND</b></p> <p>The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value 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.</p> <p><b>LBUILD_CHECK_AR</b></p> <p>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.</p> <p><b>LBUILD_VIEW</b></p> <p>Use the internal or external label representation. Value is LBUILD_VIEW_EXTERNAL by default.</p> <p><b>LBUILD_VIEW_INTERNAL</b>      Use the internal names for the highest and lowest labels in the system: ADMIN_HIGH and ADMIN_LOW.</p> <p><b>LBUILD_VIEW_EXTERNAL</b>      Promote an ADMIN_LOW label to the next highest label, and demote an ADMIN_HIGH label to the next lowest label.</p>
<b>RETURN VALUES</b>	<p>The <code>tsol_lbbuild_get()</code> returns -1 if it is unable to get the value.</p> <p>The <code>tsol_lbbuild_create()</code> routine returns a variable of type <code>ModLabelData</code> that contains the information provided in the <code>tsol_lbbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by the label builder to create the user interface.</p>
<b>EXAMPLES</b>	<p><b>EXAMPLE 1</b> To create a Label Builder</p> <pre>(ModLabelData *)lbldata = tsol_lbbuild_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);</pre> <p><b>EXAMPLE 2</b> To query the mode and display the Label Builder</p> <p>These examples call the <code>tsol_lbbuild_get()</code> routine to query the mode being used, and call the <code>tsol_lbbuild_set()</code> routine so the label builder dialog box displays.</p>

**EXAMPLE 2** To query the mode and display the Label Builder (Continued)

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

**EXAMPLE 3** To destroy the ModLabelData variable

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  
Header file for label builder functions
  - /etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

- Trusted Solaris 8 4/01 Reference Manual** label\_encodings(4)  
Trusted Solaris Developer's Guide
- SunOS 5.8 Reference Manual** attributes(5)

## labelclipping(3TSOL)

<b>NAME</b>	labelclipping, Xbcltos, Xbsltos, Xbcleartos – translate a binary label and clip to the specified width
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/label_clipping.h&gt;  XmString <b>Xbcltos</b>(Display *display, const bclabel_t *cmwlabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbsltos</b>(Display *display, const bxlabel_t *senslabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbcleartos</b>(Display *display, const bclear_t *clearance,     Dimension width, const XmFontList fontlist, const int flags);</pre>
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p><i>display</i>                      The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i>                    The CMW label to be translated.</p> <p><i>senslabel</i>                   The sensitivity label to be translated.</p> <p><i>clearance</i>                   The clearance to be translated.</p> <p><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&lt;-. 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.</p> <p><i>fontlist</i>                    A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i>                        The value of flags indicates which words in the label_encodings(4) 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].</p>
<b>RETURN VALUES</b>	These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity 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(4)* file, or not dominated by the process' sensitivity label and the PRIV\_SYS\_TRANS\_LABEL privilege is not asserted.

**FILES** /usr/dt/include/Dt/label\_clipping.h  
Header file for label clipping functions

/etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**EXAMPLES** **EXAMPLE 1** To Translate and Clip a Clearance  
  
This example translates a clearance to text using the long words specified in the *label\_encodings(4)* file, a font list, and clips the translated clearance to a width of 72 pixels.

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

**ATTRIBUTES** See *attributes(5)* for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual** *bltos(3TSOL)*, *label\_encodings(4)*  
  
*Trusted Solaris Developer's Guide*  
  
*Trusted Solaris Label Administration*

**SunOS 5.8  
Reference Manual** *attributes(5)*  
  
See *XmStringDraw(3)* and *FontList(3)* for information on the creation and structure of a font list.

labelinfo(3TSOL)

NAME	labelinfo – get information about the label encodings																		
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int labelinfo(struct label_info *info);</pre>																		
DESCRIPTION	<p>Information about the label encodings is returned in the <code>label_info</code> structure pointed to by <i>info</i>.</p> <pre>struct label_info{ short    ilabel_len;          /*obsolete */ 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 */ };</pre> <p>The fields in this structure have the following values:</p> <table><tr><td><code>ilabel_len</code></td><td>Obsolete.</td></tr><tr><td><code>slabel_len</code></td><td>The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.</td></tr><tr><td><code>clabel_len</code></td><td>The maximum length of a character-coded CMW label returned when translated from a binary CMW label.</td></tr><tr><td><code>clear_len</code></td><td>The maximum length of a character-coded clearance returned when translated from a binary clearance.</td></tr><tr><td><code>vers_len</code></td><td>The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code>.</td></tr><tr><td><code>header_len</code></td><td>The maximum length of a printer banner page header string returned by <code>bcltobanner()</code>.</td></tr><tr><td><code>protect_as_len</code></td><td>The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code>.</td></tr><tr><td><code>caveats_len</code></td><td>The maximum length of a printer banner page caveats string returned by <code>bcltobanner()</code>.</td></tr><tr><td><code>channels_len</code></td><td>The maximum length of a printer banner page channels string returned by <code>bcltobanner()</code>.</td></tr></table>	<code>ilabel_len</code>	Obsolete.	<code>slabel_len</code>	The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.	<code>clabel_len</code>	The maximum length of a character-coded CMW label returned when translated from a binary CMW label.	<code>clear_len</code>	The maximum length of a character-coded clearance returned when translated from a binary clearance.	<code>vers_len</code>	The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code> .	<code>header_len</code>	The maximum length of a printer banner page header string returned by <code>bcltobanner()</code> .	<code>protect_as_len</code>	The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code> .	<code>caveats_len</code>	The maximum length of a printer banner page caveats string returned by <code>bcltobanner()</code> .	<code>channels_len</code>	The maximum length of a printer banner page channels string returned by <code>bcltobanner()</code> .
<code>ilabel_len</code>	Obsolete.																		
<code>slabel_len</code>	The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.																		
<code>clabel_len</code>	The maximum length of a character-coded CMW label returned when translated from a binary CMW label.																		
<code>clear_len</code>	The maximum length of a character-coded clearance returned when translated from a binary clearance.																		
<code>vers_len</code>	The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code> .																		
<code>header_len</code>	The maximum length of a printer banner page header string returned by <code>bcltobanner()</code> .																		
<code>protect_as_len</code>	The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code> .																		
<code>caveats_len</code>	The maximum length of a printer banner page caveats string returned by <code>bcltobanner()</code> .																		
<code>channels_len</code>	The maximum length of a printer banner page channels string returned by <code>bcltobanner()</code> .																		
RETURN VALUES	<p><code>labelinfo()</code> returns:</p> <table><tr><td>1</td><td>On success.</td></tr><tr><td>-1</td><td>If the <code>label_encodings</code> information is unavailable.</td></tr></table>	1	On success.	-1	If the <code>label_encodings</code> information is unavailable.														
1	On success.																		
-1	If the <code>label_encodings</code> information is unavailable.																		

labelinfo(3TSOL)

**FILES** /etc/security/tsol/label\_encodings

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual** bcltobanner(3TSOL), blcompare(3TSOL), bltocolor(3TSOL), blvalid(3TSOL),  
hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual** attributes(5)

**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.

labelvers(3TSOL)

NAME	labelvers – Get version of the label_encodings file						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int labelvers(char **version, const int len);</pre>						
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 (*version[0] = '\00';).  >0       If successful, the length of the version string including the NULL terminator.						
FILES	/etc/security/tsol/label_encodings The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:  <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual	bcltobanner(3TSOL), blcompare(3TSOL), blinset(3TSOL), blmanifest(3TSOL), blminmax(3TSOL), blportion(3TSOL), bltocolour(3TSOL), bltype(3TSOL), blvalid(3TSOL), btohex(3TSOL), labelinfo(3TSOL), stobl(3TSOL), label_encodings(4)  <i>Trusted Solaris Developer's Guide</i>						
SunOS 5.8 Reference Manual	free(3C), malloc(3C), attributes(5)						
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.						



labelvers(3TSOL)

**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.

## libt6(3NSL)

<b>NAME</b>	libt6 – TSIX trusted IPC library
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -lt6 [library...] #include &lt;tsix/t6attrs.h&gt;</pre>
<b>DESCRIPTION</b>	<p>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.</p> <p>The libt6() routines provide interfaces through which the trusted application:</p> <ul style="list-style-type: none"> <li>■ 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.</li> <li>■ Controls the security options of the endpoint used to perform trusted IPC.</li> </ul>
<b>Security Attributes</b>	<p>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:</p> <p>Sensitivity Label  Nationality Caveats  Integrity Label  TSIX Session ID  Clearance  Access Control List  Effective Privileges  Audit ID  Process ID  Additional Audit Information (Deprecated. See notes.)  Additional Audit Information with Extended Address  Process Attributes  User ID  Group ID  Supplementary Group IDs</p> <p><b>Note</b> – Some of these attributes imply component security policies that may not be available on some systems. See NOTES.</p> <p>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.</p>
<b>On-Message Attribute Routines</b>	<p>The on-message attribute routines affect the security attributes associated with outgoing messages or retrieve attributes associated with incoming messages. The</p>

caller specifies attributes to these routines through a `t6attr_t` control structure (defined in `<tsix/t6attrs.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(3NSL)`

Allocates space for a `t6attr_t` control structure.

`t6free_blk(3NSL)`

Frees attribute control structure and buffers. This interface should be used in conjunction with `t6alloc_blk(3NSL)`, which allocates the space.

`t6dup_blk(3NSL)`

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(3NSL)`

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(3NSL)`

Clears attributes from a `t6attr_t` control structure.

`t6cmp_blk(3NSL)`

Compares two `t6attr_t` control structures for equal attributes set.

`t6size_attr(3NSL)`

Gets the size of an attribute from the control structure.

`t6get_attr(3NSL)`

Gets an attribute handled by the control structure.

`t6set_attr(3NSL)`

Sets an attribute handled by the control structure.

`t6sendto(3NSL)`

Sends data and a specified set of security attributes on an endpoint.

`t6recvfrom(3NSL)`

Reads a network message and retrieves the security attributes associated with the data.

`t6peek_attr(3NSL)`

Peeks ahead and returns the attributes associated with the next byte of data.

`t6last_attr(3NSL)`

Returns the security attributes associated with the last byte of data read from the network endpoint.

libt6(3NSL)

**NETWORK  
ENDPOINT  
SECURITY  
OPTIONS**

t6get\_endpt\_mask(3NSL)

Gets the endpoint mask.

t6set\_endpt\_mask(3NSL)

Sets the endpoint mask.

t6get\_endpt\_default(3NSL)

Gets the endpoint default security attributes.

t6set\_endpt\_default(3NSL)

Sets the endpoint default security attributes.

t6attr\_query(3NSL)

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(3NSL)

Turns on or off the security extensions to the network endpoint. This must be called before using any other libt6() routines.

t6new\_attr(3NSL)

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.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Trusted Solaris systems and on all other TSIX(RE) 1.1 –API compliant systems
MT-Level	MT-Safe

**TRUSTED  
SOLARIS  
SECURITY  
ATTRIBUTES**

The Trusted Solaris environment supports the following security attributes:

Sensitivity Label  
Clearance  
Effective Privileges  
Process Attributes  
Effective User ID

Effective Group ID  
Audit ID  
Additional Audit Information (Deprecated. See notes.)  
Additional Audit Information with Extended Address  
Supplemental Group IDs

The Trusted Solaris environment also supports the following attributes as read only:

Session ID  
Access Control List  
Process ID

**NOTES** This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document.

The use of the Additional Audit Information attribute is not reliable in configurations that include the use of IPv6 addresses. For this reason, the use of the Additional Audit Information attribute is deprecated. Applications needing this information should use the Additional Audit Information with Extended Address attribute.

listen(3SOCKET)

NAME	listen – listen for connections on a socket						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int <b>listen</b>(int s, int backlog);</pre>						
DESCRIPTION	<p>To accept connections, a socket is first created with <code>socket(3SOCKET)</code>, a backlog for incoming connections is specified with <code>listen()</code> and then the connections are accepted with <code>accept(3SOCKET)</code>. The <code>listen()</code> call applies only to sockets of type <code>SOCK_STREAM</code> or <code>SOCK_SEQPACKET</code>.</p> <p>The <i>backlog</i> parameter defines the maximum length the queue of pending connections may grow to.</p> <p>If a connection request arrives with the queue full, the client will receive an error with an indication of <code>ECONNREFUSED</code> for <code>AF_UNIX</code> sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For <code>AF_INET</code> 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 <code>ETIMEDOUT</code>.</p>						
RETURN VALUES	A 0 return value indicates success; -1 indicates an error. <code>listen()</code> returns:						
ERRORS	<p>The call fails if:</p> <table><tr><td><code>EBADF</code></td><td>The argument <i>s</i> is not a valid file descriptor.</td></tr><tr><td><code>ENOTSOCK</code></td><td>The argument <i>s</i> is not a socket.</td></tr><tr><td><code>EOPNOTSUPP</code></td><td>The socket is not of a type that supports the operation <code>listen()</code>.</td></tr></table>	<code>EBADF</code>	The argument <i>s</i> is not a valid file descriptor.	<code>ENOTSOCK</code>	The argument <i>s</i> is not a socket.	<code>EOPNOTSUPP</code>	The socket is not of a type that supports the operation <code>listen()</code> .
<code>EBADF</code>	The argument <i>s</i> is not a valid file descriptor.						
<code>ENOTSOCK</code>	The argument <i>s</i> is not a socket.						
<code>EOPNOTSUPP</code>	The socket is not of a type that supports the operation <code>listen()</code> .						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>If the calling process possesses the <code>PRIV_NET_MAC_READ</code> privilege, the endpoint will bind to an MLP; otherwise, it will bind to an SLP.</p> <p><code>accept(3SOCKET)</code>, <code>socket(3SOCKET)</code></p> <p><code>socket(3HEAD)</code>, <code>connect(3SOCKET)</code>, <code>attributes(5)</code></p>						
NOTES	There is currently no <i>backlog</i> limit.						

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or NULL. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to NULL.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                 /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>

## match\_execattr(3SECDB)

The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See `EXAMPLES`.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

### RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

### USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and



match\_execattr(3SECDB)

linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

## EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (\*), if defined, is returned.

```
if ((execprof=getexecprof("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

## FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/exec_attr</code>	execution profiles
<code>/etc/security/policy.conf</code>	policy definitions

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

match\_execattr(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

<b>NAME</b>	mldgetcwd – Get pathname of current working directory						
<b>SYNOPSIS</b>	<pre>cc [flags...] file... -ltsol  #include &lt;unistd.h&gt; #include &lt;tsol/mld.h&gt;  extern char *mldgetcwd(char *buf, size_t size);</pre>						
<b>DESCRIPTION</b>	<p>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.</p> <p>If <i>buf</i> is not NULL, the pathname will be stored in the space pointed to by <i>buf</i>.</p> <p>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().</p>						
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>RETURN VALUES</b>	mldgetcwd() returns NULL with errno set if <i>size</i> is not large enough, or if an error occurs in a lower-level function.						
<b>ERRORS</b>	<p>mldgetcwd() will fail if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>A parent directory cannot be read to get its name.</td></tr> <tr> <td>EINVAL</td><td><i>size</i> is equal to 0.</td></tr> <tr> <td>ERANGE</td><td><i>size</i> is greater than 0 and less than the length of the pathname plus 1.</td></tr> </table>	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.
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.						
<b>EXAMPLES</b>	<p>Here is a program that prints the current working directory.</p> <pre>#include &lt;unistd.h&gt; #include &lt;stdio.h&gt; main() {     char *cwd;     if ((cwd = mldgetcwd(NULL, 64)) == NULL) {         perror("pwd");         exit(2);     }     (void)printf("%s\n", cwd);     return(0); }</pre>						

mldgetcwd(3TSOL)

**Trusted Solaris 8**  
**4/01 Reference**  
**SunOS 5.8**  
**Reference Manual**  
**NOTES**

chdir(2)

malloc(3C), attributes(5)

Using `chdir(2)` in conjunction with `mldgetcwd()` can give unpredictable results.

<b>NAME</b>	mldstat, mldlstat – get file status in multilevel directory				
<b>SYNOPSIS</b>	<pre>#include &lt;sys/types.h&gt; #include &lt;sys/stat.h&gt;  int mldstat(const char *path, struct stat *buf); int mldlstat(const char *path, struct stat *buf);</pre>				
<b>DESCRIPTION</b>	<p>mldstat() obtains file attributes similar to those that the stat(2) system call obtains except when <i>path</i> refers to a multilevel directory (MLD); in that case, mldstat() returns information about the MLD; stat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldlstat() obtains file attributes similar to those that lstat() obtains except when the named file is an MLD; in that case, mldlstat() returns information about the MLD; lstat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldstat() and mldlstat() require mandatory read access to the final component of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, mldstat() and mldlstat() return fixed values for some elements of the stat structure.</p> <p>See the stat(2) man page for a description of the stat structure pointed to by <i>buf</i>.</p>				
<b>RETURN VALUES</b>	<p>mldstat() and mldlstat() return:</p> <p>0            On success.</p> <p>–1           On failure, and set errno to indicate the error.</p>				
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Async-Signal-Safe				
<b>NOTES</b>	<p>Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the PRIV_PROC_NODELAY privilege.</p>				
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p>link(2), stat(2)</p> <p>attributes(5), stat(5)</p>				

## mldrealpath(3TSOL)

NAME	mldrealpath, mldrealpathl – Return the canonicalized absolute pathname, including any MLD adornments and SLD names									
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;sys/param.h&gt; #include &lt;tsol/mld.h&gt;  char *mldrealpath(const char *path, char *resolved_path);  #include &lt;tsol/label.h&gt;  char *mldrealpathl(const char *path, char *resolved_path, const     bslabel_t *sl);</pre>									
DESCRIPTION	<p>mldrealpath() expands all symbolic links and resolves references to './', '../', extra '/' 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 './', '../', nor any unadorned MLDs, nor any hidden SLD names for single level directories at the present sensitivity label.</p> <p>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 does not 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.</p>									
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:									
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE									
Availability	SUNWtsu									
MT-Level	MT-Safe									
RETURN VALUES	mldrealpath() returns a pointer to the <i>resolved_path</i> on success. On failure, it returns NULL, sets <i>errno</i> 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	<table><tr><td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>.</td></tr><tr><td>EFAULT</td><td><i>resolved_path</i> extends outside the process's allocated address space.</td></tr><tr><td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr><tr><td>EINVAL</td><td><i>path</i> or <i>resolved_path</i> was NULL.</td></tr></table>		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.
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.									

	mldrealpath(3TSOL)
EIO	An I/O error occurred while reading from or writing to the file system.
ENOENT	The named file does not exist.
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).
readlink(2), getsldname(2), mldgetcwd(3TSOL)	
attributes(5)	
	It indirectly invokes the readlink(2) system call and mldgetcwd(3TSOL) library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

## mldrealpathl(3TSOL)

NAME	mldrealpath, mldrealpathl – Return the canonicalized absolute pathname, including any MLD adornments and SLD names									
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;sys/param.h&gt; #include &lt;tsol/mld.h&gt;  char *mldrealpath(const char *path, char *resolved_path);  #include &lt;tsol/label.h&gt;  char *mldrealpathl(const char *path, char *resolved_path, const     bslabel_t *sl);</pre>									
DESCRIPTION	<p>mldrealpath() expands all symbolic links and resolves references to '././', '/../.', extra '/' 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 '././', '/../.', nor any unadorned MLDs, nor any hidden SLD names for single level directories at the present sensitivity label.</p> <p>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 does not 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.</p>									
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:									
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE									
Availability	SUNWtsu									
MT-Level	MT-Safe									
RETURN VALUES	mldrealpath() returns a pointer to the <i>resolved_path</i> on success. On failure, it returns NULL, sets <i>errno</i> 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	<table><tr><td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>.</td></tr><tr><td>EFAULT</td><td><i>resolved_path</i> extends outside the process's allocated address space.</td></tr><tr><td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr><tr><td>EINVAL</td><td><i>path</i> or <i>resolved_path</i> was NULL.</td></tr></table>		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.
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.									



	mldrealpathl(3TSOL)
EIO	An I/O error occurred while reading from or writing to the file system.
ENOENT	The named file does not exist.
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).
readlink(2), getsldname(2), mldgetcwd(3TSOL)	
attributes(5)	
	It indirectly invokes the readlink(2) system call and mldgetcwd(3TSOL) library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

## mldstat(3TSOL)

<b>NAME</b>	mldstat, mldlstat – get file status in multilevel directory				
<b>SYNOPSIS</b>	<pre>#include &lt;sys/types.h&gt; #include &lt;sys/stat.h&gt;  int <b>mldstat</b>(const char *path, struct stat *buf); int <b>mldlstat</b>(const char *path, struct stat *buf);</pre>				
<b>DESCRIPTION</b>	<p>mldstat() obtains file attributes similar to those that the stat(2) system call obtains except when <i>path</i> refers to a multilevel directory (MLD); in that case, mldstat() returns information about the MLD; stat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldlstat() obtains file attributes similar to those that lstat() obtains except when the named file is an MLD; in that case, mldlstat() returns information about the MLD; lstat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldstat() and mldlstat() require mandatory read access to the final component of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, mldstat() and mldlstat() return fixed values for some elements of the stat structure.</p> <p>See the stat(2) man page for a description of the stat structure pointed to by <i>buf</i>.</p>				
<b>RETURN VALUES</b>	<p>mldstat() and mldlstat() return:</p> <p>0            On success.</p> <p>–1           On failure, and set errno to indicate the error.</p>				
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Async-Signal-Safe				
<b>NOTES</b>	<p>Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the PRIV_PROC_NODELAY privilege.</p>				
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p>link(2), stat(2)</p> <p>attributes(5), stat(5)</p>				

<b>NAME</b>	mlock, munlock – Lock or unlock pages in memory
<b>Default</b>	<pre>#include &lt;sys/mman.h&gt;  int <b>mlock</b>(caddr_t <i>addr</i>, size_t <i>len</i>);  int <b>munlock</b>(caddr_t <i>addr</i>, size_t <i>len</i>);</pre>
<b>Standard-conforming</b>	<pre>#include &lt;sys/mman.h&gt;  int <b>mlock</b>(const void * <i>addr</i>, size_t <i>len</i>);  int <b>munlock</b>(const void * <i>addr</i>, size_t <i>len</i>);</pre>
<b>DESCRIPTION</b>	<p>The <code>mlock()</code> function uses the mappings established for the address range (<i>addr</i>, <i>addr</i> + <i>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.</p> <p>The <code>munlock()</code> function removes locks established with <code>mlock()</code>.</p> <p>A given page may be locked multiple times by executing an <code>mlock()</code> 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 <code>mlock()</code> operations on the same address in the same process will all be removed with a single <code>munlock()</code>. 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.</p> <p>If the mapping through which an <code>mlock()</code> has been performed is removed, an <code>munlock()</code> is implicitly performed. An <code>munlock()</code> is also performed implicitly when a page is deleted through file removal or truncation.</p> <p>Locks established with <code>mlock()</code> are not inherited by a child process after a <code>fork()</code> and are not nested.</p> <p>Attempts to <code>mlock()</code> more memory than a system-specific limit will fail.</p>
<b>RETURN VALUES</b>	<p><code>mlock()</code> and <code>munlock()</code> return:</p> <ul style="list-style-type: none"> <li>0            On success.</li> <li>–1           On failure, and set <code>errno</code> to indicate the error. No changes are made to any locks in the address space of the process</li> </ul>
<b>ERRORS</b>	<p>The <code>mlock()</code> and <code>munlock()</code> functions will fail if:</p> <ul style="list-style-type: none"> <li>EINVAL       The <i>addr</i> argument is not a multiple of the page size as returned by <code>sysconf(3C)</code>.</li> </ul>

## mlock(3C)

ENOMEM Addresses in the range *[addr, addr + len)* are invalid for the address space of a process, or specify one or more pages which are not mapped.

ENOSYS The system does not support this memory locking interface.

EPERM The process does not have sufficient privilege.

The `mlock()` function will fail if:

EAGAIN Some or all of the memory identified by the range *[addr, addr + len)* could not be locked because of insufficient system resources.

**USAGE** 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.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF TRUSTED SQUARES CHANGES**  
To succeed, `mlock()` must have `PRIV_SYS_CONFIG` in its set of effective privileges  
`fork(2)`, `plock(3C)`, `mlockall(3C)`  
`memcntl(2)`, `mmap(2)`, `sysconf(3C)`, `attributes(5)`, `standards(5)`

<b>NAME</b>	mlockall, munlockall – Lock or unlock address space						
<b>SYNOPSIS</b>	<pre>#include &lt;sys/mman.h&gt;  int <b>mlockall</b>(int <i>flags</i>) ;  int <b>munlockall</b>(void) ;</pre>						
<b>DESCRIPTION</b>	<p>The <code>mlockall()</code> function locks in memory all pages mapped by an address space.</p> <p>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:</p> <pre> MCL_CURRENT  Lock current mappings MCL_FUTURE   Lock future mappings</pre> <p>If <code>MCL_FUTURE</code> is specified for <code>mlockall()</code>, mappings are locked as they are added to the address space (or replace existing mappings), provided sufficient memory is available. Locking in this manner is not persistent across the <code>exec</code> family of functions (see <code>exec(2)</code>).</p> <p>Mappings locked using <code>mlockall()</code> with any option may be explicitly unlocked with a <code>munlock()</code> call (see <code>mlock(3C)</code>).</p> <p>The <code>munlockall()</code> function removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory that apply to <code>mlock(3C)</code> also apply to <code>mlockall()</code>.</p> <p>Locks established with <code>mlockall()</code> are not inherited by a child process after a <code>fork(2)</code> call, and are not nested.</p>						
<b>RETURN VALUES</b>	<p><code>mlockall()</code> and <code>munlockall()</code> return:</p> <pre> 0          On success. -1         On failure, and set <code>errno</code> to indicate the error.</pre>						
<b>ERRORS</b>	<p>The <code>mlockall()</code> and <code>munlockall()</code> functions will fail if:</p> <table> <tr> <td><code>EAGAIN</code></td><td>Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.</td></tr> <tr> <td><code>EINVAL</code></td><td>The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code>.</td></tr> <tr> <td><code>EPERM</code></td><td>The process does not have sufficient privilege.</td></tr> </table>	<code>EAGAIN</code>	Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.	<code>EINVAL</code>	The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code> .	<code>EPERM</code>	The process does not have sufficient privilege.
<code>EAGAIN</code>	Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.						
<code>EINVAL</code>	The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code> .						
<code>EPERM</code>	The process does not have sufficient privilege.						
<b>USAGE</b>	The <code>mlockall()</code> and <code>munlockall()</code> functions require <code>PRIV_SYS_CONFIG</code> in their set of effective privileges.						

mlockall(3C)

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

To succeed, the mlockall() and munlockall() functions require PRIV\_SYS\_CONFIG in their set of effective privileges.

exec(2), fork(2), plock(3C), mlock(3C)

memcntl(2), mmap(2), sysconf(3C), attributes(5)

<b>NAME</b>	mlock, munlock – Lock or unlock pages in memory
<b>Default</b>	<pre>#include &lt;sys/mman.h&gt;  int mlock(caddr_t addr, size_t len);  int munlock(caddr_t addr, size_t len);</pre>
<b>Standard-conforming</b>	<pre>#include &lt;sys/mman.h&gt;  int mlock(const void * addr, size_t len);  int munlock(const void * addr, size_t len);</pre>
<b>DESCRIPTION</b>	<p>The <code>mlock()</code> function uses the mappings established for the address range (<i>addr</i>, <i>addr</i> + <i>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 <code>MAP_PRIVATE</code> page is made upon the first store, the lock will be transferred to the newly copied private page.</p> <p>The <code>munlock()</code> function removes locks established with <code>mlock()</code>.</p> <p>A given page may be locked multiple times by executing an <code>mlock()</code> 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 <code>mlock()</code> operations on the same address in the same process will all be removed with a single <code>munlock()</code>. 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.</p> <p>If the mapping through which an <code>mlock()</code> has been performed is removed, an <code>munlock()</code> is implicitly performed. An <code>munlock()</code> is also performed implicitly when a page is deleted through file removal or truncation.</p> <p>Locks established with <code>mlock()</code> are not inherited by a child process after a <code>fork()</code> and are not nested.</p> <p>Attempts to <code>mlock()</code> more memory than a system-specific limit will fail.</p>
<b>RETURN VALUES</b>	<p><code>mlock()</code> and <code>munlock()</code> return:</p> <ul style="list-style-type: none"> <li>0            On success.</li> <li>–1           On failure, and set <code>errno</code> to indicate the error. No changes are made to any locks in the address space of the process</li> </ul>
<b>ERRORS</b>	<p>The <code>mlock()</code> and <code>munlock()</code> functions will fail if:</p> <ul style="list-style-type: none"> <li><code>EINVAL</code>        The <i>addr</i> argument is not a multiple of the page size as returned by <code>sysconf(3C)</code>.</li> </ul>

## `munlock(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.
ENOSYS	The system does not support this memory locking interface.
EPERM	The process does not have sufficient privilege.
The <code>mlock()</code> function will fail if:	
EAGAIN	Some or all of the memory identified by the range <i>[addr, addr + len)</i> could not be locked because of insufficient system resources.

**USAGE** 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.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF TRUSTED SQUARES CHANGES**  
To succeed, `mlock()` must have `PRIV_SYS_CONFIG` in its set of effective privileges  
`fork(2)`, `plock(3C)`, `mlockall(3C)`  
`memcntl(2)`, `mmap(2)`, `sysconf(3C)`, `attributes(5)`, `standards(5)`



<b>NAME</b>	mlockall, munlockall – Lock or unlock address space						
<b>SYNOPSIS</b>	<pre>#include &lt;sys/mman.h&gt;  int <b>mlockall</b>(int <i>flags</i>) ;  int <b>munlockall</b>(void) ;</pre>						
<b>DESCRIPTION</b>	<p>The <code>mlockall()</code> function locks in memory all pages mapped by an address space.</p> <p>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:</p> <pre> MCL_CURRENT  Lock current mappings MCL_FUTURE   Lock future mappings</pre> <p>If <code>MCL_FUTURE</code> is specified for <code>mlockall()</code>, mappings are locked as they are added to the address space (or replace existing mappings), provided sufficient memory is available. Locking in this manner is not persistent across the <code>exec</code> family of functions (see <code>exec(2)</code>).</p> <p>Mappings locked using <code>mlockall()</code> with any option may be explicitly unlocked with a <code>munlock()</code> call (see <code>mlock(3C)</code>).</p> <p>The <code>munlockall()</code> function removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory that apply to <code>mlock(3C)</code> also apply to <code>mlockall()</code>.</p> <p>Locks established with <code>mlockall()</code> are not inherited by a child process after a <code>fork(2)</code> call, and are not nested.</p>						
<b>RETURN VALUES</b>	<p><code>mlockall()</code> and <code>munlockall()</code> return:</p> <p>0            On success.</p> <p>-1           On failure, and set <code>errno</code> to indicate the error.</p>						
<b>ERRORS</b>	<p>The <code>mlockall()</code> and <code>munlockall()</code> functions will fail if:</p> <table> <tr> <td><code>EAGAIN</code></td><td>Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.</td></tr> <tr> <td><code>EINVAL</code></td><td>The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code>.</td></tr> <tr> <td><code>EPERM</code></td><td>The process does not have sufficient privilege.</td></tr> </table>	<code>EAGAIN</code>	Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.	<code>EINVAL</code>	The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code> .	<code>EPERM</code>	The process does not have sufficient privilege.
<code>EAGAIN</code>	Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.						
<code>EINVAL</code>	The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code> .						
<code>EPERM</code>	The process does not have sufficient privilege.						
<b>USAGE</b>	The <code>mlockall()</code> and <code>munlockall()</code> functions require <code>PRIV_SYS_CONFIG</code> in their set of effective privileges.						

munlockall(3C)

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

To succeed, the mlockall() and munlockall() functions require PRIV\_SYS\_CONFIG in their set of effective privileges.

exec(2), fork(2), plock(3C), mlock(3C)

memcntl(2), mmap(2), sysconf(3C), attributes(5)

NAME	ftw, nftw – Walk a file tree																		
SYNOPSIS	<pre>#include &lt;ftw.h&gt;  int <b>ftw</b>(const char *path, int (*fn) (const char *, const struct stat     *, int), int depth);  int <b>nftw</b>(const char *path, int (*fn) (const char *, const struct     stat *, int, struct FTW*), int depth, int flags);</pre>																		
DESCRIPTION	<p>The <code>ftw()</code> function recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, <code>ftw()</code> 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 <code>stat</code> structure (see <code>stat(2)</code>) containing information about the object, and an integer. Possible values of the integer, defined in the <code>&lt;ftw.h&gt;</code> header, are:</p> <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>The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.</td></tr> </table> <p>The <code>ftw()</code> function visits a directory before visiting any of its descendants.</p> <p>The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a non-zero value, or some error is detected within <code>ftw()</code> (such as an I/O error). If the tree is exhausted, <code>ftw()</code> returns 0. If <i>fn</i> returns a non-zero value, <code>ftw()</code> stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p> <p>The <code>nftw()</code> function 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:</p> <table> <tr> <td>FTW_PHYS</td><td>Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> privileges, the walk visits all SLDs in</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	The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.	FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> privileges, the walk visits all SLDs in
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	The <code>stat()</code> function 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 <code>stat</code> buffer passed to <i>fn</i> is undefined.																		
FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> 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 <code>nftw()</code> 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 <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_SEARCH</code> 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.

The `nftw()` function calls *fn* with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the `stat` structure (see `stat(2)`) containing information about the object, the third is an integer giving additional information, and the fourth is a `struct FTW` that contains the following members:

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

The `base` member is the offset into the pathname of the base name of the object. The `level` member 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:

<code>FTW_F</code>	The object is a file.
<code>FTW_D</code>	The object is a directory.
<code>FTW_DP</code>	The object is a directory and subdirectories have been visited.
<code>FTW_SL</code>	The object is a symbolic link.
<code>FTW_SLN</code>	The object is a symbolic link that points to a non-existent file.
<code>FTW_DNR</code>	The object is a directory that cannot be read. ]The user-defined function <i>fn</i> will not be called for any of its descendants.
<code>FTW_NS</code>	The <code>stat()</code> function failed on the object because of lack of appropriate permission. The <code>stat</code> buffer passed to <i>fn</i> is undefined. The <code>stat()</code> function failed for a reason other than lack of appropriate permission. <code>EACCES</code> 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. It must not be greater than the number of file descriptors currently available for use. The ftw() function 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 the tree is exhausted, ftw() and nftw() return 0. If the function pointed to by *fn* returns a non-zero value, ftw() and nftw() stop their tree traversal and return whatever value was returned by the function pointed to by *fn*. If ftw() and nftw() detect an error, they return -1 and set *errno* to indicate the error.

If ftw() and nftw() encounter an error other than EACCES (see FTW\_DNR and FTW\_NS above), they return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error value that is possible when a directory is opened or when one of the stat functions is executed on a directory or file.

## ERRORS

The ftw() and nftw() functions will fail if:

ENAMETOOLONG	The length of the path exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENOTDIR	A component of <i>path</i> is not a directory.

The ftw() function will fail if:

EACCES	Search permission is denied for any component of <i>path</i> or read permission is denied for <i>path</i> .
ELOOP	Too many symbolic links were encountered.

The nftw() function will fail if:

EACCES	Search permission is denied for any component of <i>path</i> or read permission is denied for <i>path</i> , or <i>fn</i> () returns -1 and does not reset <i>errno</i> .
--------	--

The ftw() and nftw() functions may fail if:

ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
--------------	---

## nftw(3C)

The `ftw()` function may fail if:

`EINVAL` The value of the `ndirs` argument is invalid.

The `nftw()` function may fail if:

`ELOOP` Too many symbolic links were encountered in resolving *path*.

`EMFILE` There are `OPEN_MAX` file descriptors currently open in the calling process.

`ENFILE` Too many files are currently open in the system.

In addition, if the function pointed to by *fn* encounters system errors, `errno` may be set accordingly.

### USAGE

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

The `ftw()` function 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 non-zero value at its next invocation.

The `ftw()` and `nftw()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

The `ftw()` function is safe in multithreaded applications. The `nftw()` function is safe in multithreaded 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`.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe with exceptions.

### 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.

### Trusted Solaris 8 4/01 Reference Manual

`stat(2)`



## nis\_add(3NSL)

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *nis_lookup(nis_name name, uint_t flags); nis_result *nis_add(nis_name name, nis_object *obj); nis_result *nis_remove(nis_name name, nis_object *obj); nis_result *nis_modify(nis_name name, nis_object *obj); void nis_freeresult(nis_result *result);</pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table> <tr> <td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr> </table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	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 <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

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

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.

nis\_add(3NSL)

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.

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj    cookie;
    uint32_t  zticks;
    uint32_t  dticks;
    uint32_t  aticks;
    uint32_t  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(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

`nis_add(3NSL)`

*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.

## 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 *expired*. 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 *add*, *remove*, and *modify* 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\_add(3NSL)

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 *obj* 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 *LINK* 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(3C)* message indicating why the RPC request failed.



## nis\_add\_entry(3NSL)

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *nis_list(nis_name name, uint_t flags, int     (*callback)(nis_name table_name, nis_object *object, void     *userdata), void *userdata);  nis_result *nis_add_entry(nis_name table_name, nis_object *object,     uint_t flags);  nis_result *nis_remove_entry(nis_name name, nis_object *object,     uint_t flags);  nis_result *nis_modify_entry(nis_name name, nis_object *object,     uint_t flags);  nis_result *nis_first_entry(nis_name table_name);  nis_result *nis_next_entry(nis_name table_name, netobj *cookie);  void nis_freeresult(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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

## `nis_add_entry(3NSL)`

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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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



`nis_add_entry(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    uint32_t     zticks;
    uint32_t     dticks;
    uint32_t     aticks;
    uint32_t     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(3NSL)`].

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(3NSL)`]). 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(3NSL)`).

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.

## nis\_add\_entry(3NSL)

<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 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.
<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.

## 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 *expired*. 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

`nis_add_entry(3NSL)`

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 *object* 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()`.

## `nis_add_entry(3NSL)`

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(3C)` 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(3C)` 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

	<code>nis_add_entry(3NSL)</code>				
	server is currently updating its internal state. It can be returned to <code>nis_list()</code> when the function specifies a callback and the server does not have the resources to handle callbacks.				
	<p><code>NIS_TYPEMISMATCH</code></p> <p>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.</p>				
ENVIRONMENT VARIABLES	<p><code>NIS_PATH</code> When set, this variable is the search path used by <code>nis_list()</code> if the flag <code>EXPAND_NAME</code> is set.</p>				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe with exceptions				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual  SunOS 5.8 Reference Manual	<p>To succeed, <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_cachemgr(1M)</code>, <code>nis_names(3NSL)</code>, <code>nis_server(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code></p> <p><code>niscat(1)</code>, <code>niserror(1)</code>, <code>nismatch(1)</code>, <code>syslog(3C)</code>, <code>nis_clone_object(3NSL)</code>, <code>nis_destroy_object(3NSL)</code>, <code>nis_error(3NSL)</code>, <code>nis_getnames(3NSL)</code>, <code>nis_local_names(3NSL)</code>, <code>nis_objects(3NSL)</code>, <code>attributes(5)</code></p>				
WARNINGS	Use the flag <code>HARD_LOOKUP</code> carefully since it can cause the application to block indefinitely during a network partition.				
NOTES	<p>The path used when the flag <code>FOLLOW_PATH</code> is specified, is the one present in the <i>first</i> table searched. The path values in tables that are subsequently searched are ignored.</p> <p>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 <code>nis_list()</code> with a callback from within a list callback function is not currently supported.</p> <p>There are currently no known methods for <code>nis_first_entry()</code> and <code>nis_next_entry()</code> to get their answers from only the master server.</p> <p>The <code>nis_list()</code> function is not MT-Safe with callbacks. <code>nis_list()</code> callbacks are serialized. A call to <code>nis_list()</code> with a callback from within <code>nis_list()</code> will deadlock. <code>nis_list()</code> with a callback cannot be called from an rpc server. See <code>rpc_svc_calls(3NSL)</code>. Otherwise, this function is MT-Safe.</p>				

nis\_addmember(3NSL)

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ;  nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ;  nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ;  nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ;  nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ;  void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ;  nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

`nis_addmember(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

### EXAMPLE 1 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).
```

### EXAMPLE 2 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).
```

### EXAMPLE 3 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.
```

## ATTRIBUTES

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

nis\_addmember(3NSL)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
SunOS 5.8  
Reference Manual  
NOTES**

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;    /* Interpretation Flags
                        (currently unused) */
struct {
    uint_t    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	nis_ping, nis_checkpoint – Misc NIS+ log administration functions				
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  void <b>nis_ping</b>(nis_name <i>dirname</i>, uint32_t <i>utime</i>, nis_object *<i>diobj</i>) ;  nis_result *<b>nis_checkpoint</b>(nis_name <i>dirname</i>) ;</pre>				
DESCRIPTION	<p><b>nis_ping()</b> 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>diobj</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.</p> <p>The effect of calling <b>nis_ping()</b> 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.</p> <p>To succeed, <b>nis_ping()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p><b>nis_checkpoint()</b> 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.</p> <p><b>nis_checkpoint()</b> returns a pointer to a <i>nis_result</i> structure (described in <b>nis_tables(3NSL)</b>). This structure should be freed with <b>nis_freeresult()</b> (see <b>nis_names(3NSL)</b>). The only items of interest in the returned result are the status value and the statistics.</p>				
ATTRIBUTES	<p>See <b>attributes(5)</b> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SUMMARY OF TRUSTED SQUARIS Trusted Squares 4/01 Changes Manual	<p>To succeed, <b>nis_ping()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p><b>nis_names(3NSL)</b>, <b>nis_tables(3NSL)</b></p>				

nis\_checkpoint(3NSL)

**SunOS 5.8** nislog(1M), nisfiles(4), attributes(5)  
**Reference Manual**

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

## `nis_creategroup(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

#### EXAMPLE 1 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).
```

#### EXAMPLE 2 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).
```

#### EXAMPLE 3 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.
```

### ATTRIBUTES

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

`nis_creategroup(3NSL)`

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;      /* Interpretation Flags
                          (currently unused) */
struct {
    uint_t    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.

## nis\_destroygroup(3NSL)

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"><li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li><li>■ 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.</li><li>■ 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.</li></ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

`nis_destroygroup(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

### EXAMPLE 1 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).
```

### EXAMPLE 2 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).
```

### EXAMPLE 3 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.
```

## ATTRIBUTES

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

nis\_destroygroup(3NSL)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
SunOS 5.8  
Reference Manual  
NOTES**

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;    /* Interpretation Flags
                        (currently unused) */
struct {
    uint_t    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.



nis\_first\_entry(3NSL)

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre><b>cc</b> [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_list</b>(nis_name name, uint_t flags, int     (*callback)(nis_name table_name, nis_object *object, void     *userdata), void *userdata);  nis_result *<b>nis_add_entry</b>(nis_name table_name, nis_object *object,     uint_t flags);  nis_result *<b>nis_remove_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_modify_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_first_entry</b>(nis_name table_name);  nis_result *<b>nis_next_entry</b>(nis_name table_name, netobj *cookie);  void <b>nis_freeresult</b>(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, 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 <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

## nis\_first\_entry(3NSL)

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

## `nis_first_entry(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj      cookie;
    uint32_t     zticks;
    uint32_t     dticks;
    uint32_t     aticks;
    uint32_t     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(3NSL)`].

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(3NSL)`). 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(3NSL)`).

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.

nis\_first\_entry(3NSL)

<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 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.
<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.

## 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 *expired*. 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

`nis_first_entry(3NSL)`

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 *object* 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(3C)` 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(3C)` 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

`nis_first_entry(3NSL)`

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  
VARIABLES**

**NIS\_PATH** When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3NSL)`, `nis_server(3NSL)`,  
`rpc_svc_calls(3NSL)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3C)`, `nis_clone_object(3NSL)`,  
`nis_destroy_object(3NSL)`, `nis_error(3NSL)`, `nis_getnames(3NSL)`,  
`nis_local_names(3NSL)`, `nis_objects(3NSL)`, `attributes(5)`

**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.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3NSL)`. Otherwise, this function is MT-Safe.



NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre> <b>cc</b> [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_lookup</b>(nis_name name, uint_t flags); nis_result *<b>nis_add</b>(nis_name name, nis_object *obj); nis_result *<b>nis_remove</b>(nis_name name, nis_object *obj); nis_result *<b>nis_modify</b>(nis_name name, nis_object *obj); void <b>nis_freeresult</b>(nis_result *result); </pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table> <tr> <td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr> </table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the		

## `nis_freeresult(3NSL)`

	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.
<code>HARD_LOOKUP</code>	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.
<code>NO_CACHE</code>	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.
<code>MASTER_ONLY</code>	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.
<code>EXPAND_NAME</code>	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperno()` (see `nis_error(3NSL)`).

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.

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 {
        uint_t      objects_len;
        nis_object  *objects_val;
    } objects;
    netobj      cookie;
    uint32_t     zticks;
    uint32_t     dticks;
    uint32_t     aticks;
    uint32_t     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_strerror()` (see `nis_error(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

nis\_freeresult(3NSL)

*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.

## 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 *expired*. 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 *add*, *remove*, and *modify* 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 *obj* 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 *LINK* 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(3C)` message indicating why the RPC request failed.

nis\_freeresult(3NSL)

**ENVIRONMENT  
VARIABLES**

NIS\_PATH            If the flag EXPAND\_NAME is set, this variable is the search path  
                      used by nis\_lookup().

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

Trusted Solaris 8  
4/01 Reference  
Manual  
Sun Microsystems  
Reference Manual

To succeed, nis\_add(), nis\_modify(), and nis\_remove() must inherit the  
PAF\_TRUSTED\_PATH attribute.

nis\_server(3NSL), nis\_tables(3NSL)

nis\_error(3NSL), nis\_objects(3NSL), nis\_subr(3NSL), attributes(5)

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_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freesservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_freetags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freesservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

## nis\_freesservlist(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`.

`nis_freesservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`



NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_frehtags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_frehtags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freeservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

nis\_freetags(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference SunOS 5.8 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_error nis_mkdir(nis_name dirname, nis_server *machine); nis_error nis_rmdir(nis_name dirname, nis_server *machine); nis_error nis_servstate(nis_server *machine, nis_tag *tags, int     numtags, nis_tag **result); nis_error nis_stats(nis_server *machine, nis_tag *tags, int numtags,     nis_tag **result); void nis_freetags(nis_tag *tags, int numtags); nis_server **nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server **machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> 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 <code>nis_server</code> structure, refer to <code>nis_objects(3NSL)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>Per-server and per-directory access restrictions may apply to <code>nis_mkdir()</code>. See <code>nisopaccess(1)</code>.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <code>nis_rmdir()</code> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <code>nis_remove()</code> to remove the directory <i>dirname</i> from the namespace and call <code>nis_rmdir()</code> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <code>nis_modify()</code> to remove the server from the directory object and then call <code>nis_rmdir()</code> to remove the replica. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>Per-server and per-directory access restrictions may apply to <code>nis_rmdir()</code>. See <code>nisopaccess(1)</code>.</p> <p><code>nis_servstate()</code> 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, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p>

nis\_getservlist(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference SunOS 5.8 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

## `nis_groups(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

#### EXAMPLE 1 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).
```

#### EXAMPLE 2 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).
```

#### EXAMPLE 3 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.
```

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;      /* Interpretation Flags
                          (currently unused) */
struct {
    uint_t  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.

## nis\_ismember(3NSL)

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>



`nis_ismember(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

### EXAMPLE 1 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).
```

### EXAMPLE 2 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).
```

### EXAMPLE 3 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.
```

## ATTRIBUTES

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

`nis_ismember(3NSL)`

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
SunOS 5.8  
Reference Manual  
NOTES**

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;    /* Interpretation Flags
                        (currently unused) */
struct {
    uint_t    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	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_list</b>(nis_name name, uint_t flags, int     (*callback)(nis_name table_name, nis_object *object, void     *userdata), void *userdata);  nis_result *<b>nis_add_entry</b>(nis_name table_name, nis_object *object,     uint_t flags);  nis_result *<b>nis_remove_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_modify_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_first_entry</b>(nis_name table_name);  nis_result *<b>nis_next_entry</b>(nis_name table_name, netobj *cookie);  void <b>nis_freeresult</b>(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, 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 <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

## nis\_list(3NSL)

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

`nis_list(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj       cookie;
    uint32_t      zticks;
    uint32_t      dticks;
    uint32_t      aticks;
    uint32_t      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(3NSL)`].

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(3NSL)`]). 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(3NSL)`).

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.

<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 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.
<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.

## 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 *expired*. 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

`nis_list(3NSL)`

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 *object* 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(3C) 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(3C) 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

`nis_list(3NSL)`

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  
VARIABLES**

**NIS\_PATH** When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3NSL)`, `nis_server(3NSL)`,  
`rpc_svc_calls(3NSL)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3C)`, `nis_clone_object(3NSL)`,  
`nis_destroy_object(3NSL)`, `nis_error(3NSL)`, `nis_getnames(3NSL)`,  
`nis_local_names(3NSL)`, `nis_objects(3NSL)`, `attributes(5)`

**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.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3NSL)`. Otherwise, this function is MT-Safe.

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_lookup</b>(nis_name name, uint_t flags); nis_result *<b>nis_add</b>(nis_name name, nis_object *obj); nis_result *<b>nis_remove</b>(nis_name name, nis_object *obj); nis_result *<b>nis_modify</b>(nis_name name, nis_object *obj); void <b>nis_freeresult</b>(nis_result *result);</pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table> <tr> <td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr> </table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the		

## nis\_lookup(3NSL)

	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 <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

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

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.

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 {
        uint_t      objects_len;
        nis_object  *objects_val;
    } objects;
    netobj      cookie;
    uint32_t     zticks;
    uint32_t     dticks;
    uint32_t     aticks;
    uint32_t     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_strerror()` (see `nis_error(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

## nis\_lookup(3NSL)

*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.

### 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 *expired*. 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 *add*, *remove*, and *modify* 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 *obj* 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 *LINK* 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*(3C) message indicating why the RPC request failed.

nis\_lookup(3NSL)

**ENVIRONMENT  
VARIABLES**

NIS\_PATH            If the flag EXPAND\_NAME is set, this variable is the search path  
used by nis\_lookup().

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

Trusted Solaris 8  
4/01 Reference  
Manual  
Sun Microsystems  
Reference Manual

To succeed, nis\_add(), nis\_modify(), and nis\_remove() must inherit the  
PAF\_TRUSTED\_PATH attribute.

nis\_server(3NSL), nis\_tables(3NSL)

nis\_error(3NSL), nis\_objects(3NSL), nis\_subr(3NSL), attributes(5)

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_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_error nis_mkdir(nis_name dirname, nis_server *machine); nis_error nis_rmdir(nis_name dirname, nis_server *machine); nis_error nis_servstate(nis_server *machine, nis_tag *tags, int     numtags, nis_tag **result); nis_error nis_stats(nis_server *machine, nis_tag *tags, int numtags,     nis_tag **result); void nis_freetags(nis_tag *tags, int numtags); nis_server **nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server **machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> 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 <code>nis_server</code> structure, refer to <code>nis_objects(3NSL)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>Per-server and per-directory access restrictions may apply to <code>nis_mkdir()</code>. See <code>nisopaccess(1)</code>.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <code>nis_rmdir()</code> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <code>nis_remove()</code> to remove the directory <i>dirname</i> from the namespace and call <code>nis_rmdir()</code> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <code>nis_modify()</code> to remove the server from the directory object and then call <code>nis_rmdir()</code> to remove the replica. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>Per-server and per-directory access restrictions may apply to <code>nis_rmdir()</code>. See <code>nisopaccess(1)</code>.</p> <p><code>nis_servstate()</code> 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, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p>

## nis\_mkdir(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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_frehtags()` to free the tag structure.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference SunOS 5.8 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_lookup</b>(nis_name name, uint_t flags); nis_result *<b>nis_add</b>(nis_name name, nis_object *obj); nis_result *<b>nis_remove</b>(nis_name name, nis_object *obj); nis_result *<b>nis_modify</b>(nis_name name, nis_object *obj); void <b>nis_freeresult</b>(nis_result *result);</pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table> <tr> <td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr> </table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the		

## `nis_modify(3NSL)`

	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.
<code>HARD_LOOKUP</code>	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.
<code>NO_CACHE</code>	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.
<code>MASTER_ONLY</code>	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.
<code>EXPAND_NAME</code>	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

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

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.

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 {
        uint_t      objects_len;
        nis_object   *objects_val;
    } objects;
    netobj    cookie;
    uint32_t   zticks;
    uint32_t   dticks;
    uint32_t   aticks;
    uint32_t   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_strerror()` (see `nis_strerror(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

nis\_modify(3NSL)

*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.

## 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 *expired*. 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 *add*, *remove*, and *modify* 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 *obj* 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 *LINK* 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(3C)` message indicating why the RPC request failed.

nis\_modify(3NSL)

**ENVIRONMENT  
VARIABLES**

NIS\_PATH            If the flag EXPAND\_NAME is set, this variable is the search path  
                      used by nis\_lookup().

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

To succeed, nis\_add(), nis\_modify(), and nis\_remove() must inherit the  
PAF\_TRUSTED\_PATH attribute.

nis\_server(3NSL), nis\_tables(3NSL)

nis\_error(3NSL), nis\_objects(3NSL), nis\_subr(3NSL), attributes(5)

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_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_list</b>(nis_name <i>name</i>, uint_t <i>flags</i>, int     (*callback)(nis_name <i>table_name</i>, nis_object *<i>object</i>, void     *<i>userdata</i>), void *<i>userdata</i>);  nis_result *<b>nis_add_entry</b>(nis_name <i>table_name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_remove_entry</b>(nis_name <i>name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_modify_entry</b>(nis_name <i>name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_first_entry</b>(nis_name <i>table_name</i>);  nis_result *<b>nis_next_entry</b>(nis_name <i>table_name</i>, netobj *<i>cookie</i>);  void <b>nis_freeresult</b>(nis_result *<i>result</i>); </pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

## nis\_modify\_entry(3NSL)

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

## `nis_modify_entry(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj       cookie;
    uint32_t     zticks;
    uint32_t     dticks;
    uint32_t     aticks;
    uint32_t     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(3NSL)`].

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(3NSL)`]). 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(3NSL)`).

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.

<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 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.
<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.

<b>ERRORS</b>	The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.
	<b>NIS_BADATTRIBUTE</b> 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.
	<b>NIS_BADNAME</b> The name passed to the function is not a legal NIS+ name.
	<b>NIS_BADREQUEST</b> A problem was detected in the request structure passed to the client library.
	<b>NIS_CACHEEXPIRED</b> 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 <b>NO_CACHE</b> was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.
	<b>NIS_CBERROR</b> 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.
	<b>NIS_CBRESULTS</b> 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.
	<b>NIS_FOREIGNNS</b> 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

## `nis_modify_entry(3NSL)`

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 *object* 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()`.

## `nis_modify_entry(3NSL)`

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(3C)` 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(3C)` 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

## `nis_modify_entry(3NSL)`

### ENVIRONMENT VARIABLES

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.

### ATTRIBUTES

`NIS_PATH` When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3NSL)`, `nis_server(3NSL)`,  
`rpc_svc_calls(3NSL)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3C)`, `nis_clone_object(3NSL)`,  
`nis_destroy_object(3NSL)`, `nis_error(3NSL)`, `nis_getnames(3NSL)`,  
`nis_local_names(3NSL)`, `nis_objects(3NSL)`, `attributes(5)`

### 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.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3NSL)`. Otherwise, this function is MT-Safe.



NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_lookup</b>(nis_name name, uint_t flags); nis_result *<b>nis_add</b>(nis_name name, nis_object *obj); nis_result *<b>nis_remove</b>(nis_name name, nis_object *obj); nis_result *<b>nis_modify</b>(nis_name name, nis_object *obj); void <b>nis_freeresult</b>(nis_result *result);</pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table> <tr> <td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr> </table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the		

## nis\_names(3NSL)

	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 <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

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

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.

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj    cookie;
    uint32_t  zticks;
    uint32_t  dticks;
    uint32_t  aticks;
    uint32_t  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_strerror()` (see `nis_error(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

nis\_names(3NSL)

*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.

## 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 *expired*. 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 *add*, *remove*, and *modify* 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 *obj* 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 *LINK* 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(3C)` message indicating why the RPC request failed.

nis\_names(3NSL)

## ENVIRONMENT VARIABLES

NIS\_PATH            If the flag EXPAND\_NAME is set, this variable is the search path used by nis\_lookup().

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual NOTES

To succeed, nis\_add(), nis\_modify(), and nis\_remove() must inherit the PAF\_TRUSTED\_PATH attribute.

nis\_server(3NSL), nis\_tables(3NSL)

nis\_error(3NSL), nis\_objects(3NSL), nis\_subr(3NSL), attributes(5)

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.

nis\_next\_entry(3NSL)

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre><b>cc</b> [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_list</b>(nis_name name, uint_t flags, int     (*callback)(nis_name table_name, nis_object *object, void     *userdata), void *userdata);  nis_result *<b>nis_add_entry</b>(nis_name table_name, nis_object *object,     uint_t flags);  nis_result *<b>nis_remove_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_modify_entry</b>(nis_name name, nis_object *object,     uint_t flags);  nis_result *<b>nis_first_entry</b>(nis_name table_name);  nis_result *<b>nis_next_entry</b>(nis_name table_name, netobj *cookie);  void <b>nis_freeresult</b>(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, 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 <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

## nis\_next\_entry(3NSL)

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

## `nis_next_entry(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj       cookie;
    uint32_t      zticks;
    uint32_t      dticks;
    uint32_t      aticks;
    uint32_t      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(3NSL)`].

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(3NSL)`]). 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(3NSL)`).

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.

nis\_next\_entry(3NSL)

<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 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.
<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.

## 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 *expired*. 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

`nis_next_entry(3NSL)`

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 *object* 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(3C)` 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(3C)` 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

`nis_next_entry(3NSL)`

**ENVIRONMENT  
VARIABLES**

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.

**ATTRIBUTES**

**NIS\_PATH** When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3NSL)`, `nis_server(3NSL)`,  
`rpc_svc_calls(3NSL)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3C)`, `nis_clone_object(3NSL)`,  
`nis_destroy_object(3NSL)`, `nis_error(3NSL)`, `nis_getnames(3NSL)`,  
`nis_local_names(3NSL)`, `nis_objects(3NSL)`, `attributes(5)`

**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.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3NSL)`. Otherwise, this function is MT-Safe.

NAME	nis_ping, nis_checkpoint – Misc NIS+ log administration functions				
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  void <b>nis_ping</b>(nis_name <i>dirname</i>, uint32_t <i>utime</i>, nis_object *<i>diobj</i>) ;  nis_result *<b>nis_checkpoint</b>(nis_name <i>dirname</i>) ;</pre>				
DESCRIPTION	<p><b>nis_ping()</b> 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>diobj</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.</p> <p>The effect of calling <b>nis_ping()</b> 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.</p> <p>To succeed, <b>nis_ping()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p><b>nis_checkpoint()</b> 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.</p> <p><b>nis_checkpoint()</b> returns a pointer to a <i>nis_result</i> structure (described in <b>nis_tables(3NSL)</b>). This structure should be freed with <b>nis_freeresult()</b> (see <b>nis_names(3NSL)</b>). The only items of interest in the returned result are the status value and the statistics.</p>				
ATTRIBUTES	<p>See <b>attributes(5)</b> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SUMMARY OF TRUSTED SQUARIS Trusted Squares 4/01 Changes Manual	<p>To succeed, <b>nis_ping()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p><b>nis_names(3NSL)</b>, <b>nis_tables(3NSL)</b></p>				

nis\_ping(3NSL)

**SunOS 5.8  
Reference Manual**

nislog(1M), nisfiles(4), attributes(5)



nis\_print\_group\_entry(3NSL)

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t nis_ismember(nis_name principal, nis_name group); nis_error nis_addmember(nis_name member, nis_name group); nis_error nis_removemember(nis_name member, nis_name group); nis_error nis_creategroup(nis_name group, uint_t flags); nis_error nis_destroygroup(nis_name group); void nis_print_group_entry(nis_name group); nis_error nis_verifygroup(nis_name group);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"><li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li><li>■ 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.</li><li>■ 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.</li></ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

`nis_print_group_entry(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

### EXAMPLE 1 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).
```

### EXAMPLE 2 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).
```

### EXAMPLE 3 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.
```

## ATTRIBUTES

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

nis\_print\_group\_entry(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;      /* Interpretation Flags
                          (currently unused) */
struct {
    uint_t    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.

nis\_remove(3NSL)

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions		
SYNOPSIS	<pre><b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_lookup</b>(nis_name <i>name</i>, uint_t <i>flags</i>); nis_result *<b>nis_add</b>(nis_name <i>name</i>, nis_object *<i>obj</i>); nis_result *<b>nis_remove</b>(nis_name <i>name</i>, nis_object *<i>obj</i>); nis_result *<b>nis_modify</b>(nis_name <i>name</i>, nis_object *<i>obj</i>); void <b>nis_freeresult</b>(nis_result *<i>result</i>);</pre>		
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3NSL)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3NSL)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>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 <code>nis_subr(3NSL)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> 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 <code>nis_clone_object(3NSL)</code> (see <code>nis_subr(3NSL)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> 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 <code>nis_tables(3NSL)</code>). The <code>nis_lookup()</code> 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 <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically ORing zero or more flags from the following list.</p> <table><tr><td><code>FOLLOW_LINKS</code></td><td>When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the</td></tr></table>	<code>FOLLOW_LINKS</code>	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the
<code>FOLLOW_LINKS</code>	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 <code>nis_getnames()</code> (see <code>nis_subr(3NSL)</code> ) which uses the environment variable <code>NIS_PATH</code> .

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

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.

## nis\_remove(3NSL)

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.

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj    cookie;
    uint32_t  zticks;
    uint32_t  dticks;
    uint32_t  aticks;
    uint32_t  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(3NSL)`).

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(3NSL)`). Refer to `nis_objects(3NSL)` 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.

<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.

`nis_remove(3NSL)`

*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.

## 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 *expired*. 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 *add*, *remove*, and *modify* 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_remove(3NSL)`

### `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 *obj* 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 *LINK* 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(3C)` message indicating why the RPC request failed.





## nis\_remove\_entry(3NSL)

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_result *nis_list(nis_name name, uint_t flags, int     (*callback)(nis_name table_name, nis_object *object, void     *userdata), void *userdata);  nis_result *nis_add_entry(nis_name table_name, nis_object *object,     uint_t flags);  nis_result *nis_remove_entry(nis_name name, nis_object *object,     uint_t flags);  nis_result *nis_modify_entry(nis_name name, nis_object *object,     uint_t flags);  nis_result *nis_first_entry(nis_name table_name);  nis_result *nis_next_entry(nis_name table_name, netobj *cookie);  void nis_freeresult(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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

## `nis_remove_entry(3NSL)`

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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

`nis_remove_entry(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj    cookie;
    uint32_t    zticks;
    uint32_t    dticks;
    uint32_t    aticks;
    uint32_t    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(3NSL)`].

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(3NSL)`]). 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(3NSL)`).

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.

nis\_remove\_entry(3NSL)

<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 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.
<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.

## 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 *expired*. 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

`nis_remove_entry(3NSL)`

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 *object* 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()`.

## `nis_remove_entry(3NSL)`

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(3C)` 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(3C)` 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



	<code>nis_remove_entry(3NSL)</code>				
	server is currently updating its internal state. It can be returned to <code>nis_list()</code> when the function specifies a callback and the server does not have the resources to handle callbacks.				
	<p><code>NIS_TYPEMISMATCH</code></p> <p>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.</p>				
ENVIRONMENT VARIABLES	<p><code>NIS_PATH</code> When set, this variable is the search path used by <code>nis_list()</code> if the flag <code>EXPAND_NAME</code> is set.</p>				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe with exceptions				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual  SunOS 5.8 Reference Manual	<p>To succeed, <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_cachemgr(1M)</code>, <code>nis_names(3NSL)</code>, <code>nis_server(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code></p> <p><code>niscat(1)</code>, <code>niserror(1)</code>, <code>nismatch(1)</code>, <code>syslog(3C)</code>, <code>nis_clone_object(3NSL)</code>, <code>nis_destroy_object(3NSL)</code>, <code>nis_error(3NSL)</code>, <code>nis_getnames(3NSL)</code>, <code>nis_local_names(3NSL)</code>, <code>nis_objects(3NSL)</code>, <code>attributes(5)</code></p>				
WARNINGS	Use the flag <code>HARD_LOOKUP</code> carefully since it can cause the application to block indefinitely during a network partition.				
NOTES	<p>The path used when the flag <code>FOLLOW_PATH</code> is specified, is the one present in the <i>first</i> table searched. The path values in tables that are subsequently searched are ignored.</p> <p>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 <code>nis_list()</code> with a callback from within a list callback function is not currently supported.</p> <p>There are currently no known methods for <code>nis_first_entry()</code> and <code>nis_next_entry()</code> to get their answers from only the master server.</p> <p>The <code>nis_list()</code> function is not MT-Safe with callbacks. <code>nis_list()</code> callbacks are serialized. A call to <code>nis_list()</code> with a callback from within <code>nis_list()</code> will deadlock. <code>nis_list()</code> with a callback cannot be called from an rpc server. See <code>rpc_svc_calls(3NSL)</code>. Otherwise, this function is MT-Safe.</p>				

## nis\_removemember(3NSL)

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

`nis_removemember(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

### EXAMPLE 1 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).
```

### EXAMPLE 2 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).
```

### EXAMPLE 3 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.
```

## ATTRIBUTES

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

nis\_removemember(3NSL)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES  
SunOS 5.8  
Reference Manual  
NOTES**

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;    /* Interpretation Flags
                        (currently unused) */
struct {
    uint_t    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	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_freetags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freeservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

## nis\_rmdir(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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_frehtags()` to free the tag structure.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference SunOS 5.8 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_freetags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freeservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

nis\_server(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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_frehtags()` to free the tag structure.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`



NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_freetags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freeservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b></p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

nis\_servstate(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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_frehtags()` to free the tag structure.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre> <b>cc</b> [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  nis_error <b>nis_mkdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_rmdir</b>(nis_name <i>dirname</i>, nis_server *<i>machine</i>) ; nis_error <b>nis_servstate</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int     <i>numtags</i>, nis_tag **<i>result</i>) ; nis_error <b>nis_stats</b>(nis_server *<i>machine</i>, nis_tag *<i>tags</i>, int <i>numtags</i>,     nis_tag **<i>result</i>) ; void <b>nis_freetags</b>(nis_tag *<i>tags</i>, int <i>numtags</i>) ; nis_server **<b>nis_getservlist</b>(nis_name <i>dirname</i>) ; void <b>nis_freeservlist</b>(nis_server **<i>machines</i>) ; </pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><b>nis_mkdir()</b> 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 <b>nis_server</b> structure, refer to <b>nis_objects(3NSL)</b>. To succeed, <b>nis_mkdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_mkdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_rmdir()</b> is used to delete the directory, <i>dirname</i>, from the specified server machine. The <i>machine</i> parameter cannot be NULL. Note that <b>nis_rmdir()</b> does not remove the directory <i>dirname</i> from the namespace or remove a server from the server list in the directory object. To remove a directory from the namespace you must call <b>nis_remove()</b> to remove the directory <i>dirname</i> from the namespace and call <b>nis_rmdir()</b> for each server in the server list to remove the directory from the server. To remove a replica from the server list, you need to first call <b>nis_modify()</b> to remove the server from the directory object and then call <b>nis_rmdir()</b> to remove the replica. To succeed, <b>nis_rmdir()</b> must inherit the PAF_TRUSTED_PATH attribute.</p> <p>Per-server and per-directory access restrictions may apply to <b>nis_rmdir()</b>. See <b>nisopaccess(1)</b>.</p> <p><b>nis_servstate()</b> 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, <b>nis_servstate()</b> must inherit the PAF_TRUSTED_PATH attribute.</p>

## nis\_stats(3NSL)

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 `nis_tag` structures whose length is passed to the function in the `numtags` 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 `tag_value` 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.

Per-server and per-directory access restrictions may apply to the `NIS_SERVSTATE` or `NIS_STATUS(nis_stats())` operations and their sub-operations (*tags*). See `nisopaccess(1)`

`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(3NSL)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference SunOS 5.8 Reference Manual

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3NSL)`

`nisopaccess(1)`, `nisstat(1M)`, `nis_objects(3NSL)`, `nis_subr(3NSL)`,  
`attributes(5)`

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lnsl [<i>library...</i>]  #include &lt;rpcsvc/nis.h&gt;  nis_result *<b>nis_list</b>(nis_name <i>name</i>, uint_t <i>flags</i>, int     (*callback)(nis_name <i>table_name</i>, nis_object *<i>object</i>, void     *<i>userdata</i>), void *<i>userdata</i>);  nis_result *<b>nis_add_entry</b>(nis_name <i>table_name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_remove_entry</b>(nis_name <i>name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_modify_entry</b>(nis_name <i>name</i>, nis_object *<i>object</i>,     uint_t <i>flags</i>);  nis_result *<b>nis_first_entry</b>(nis_name <i>table_name</i>);  nis_result *<b>nis_next_entry</b>(nis_name <i>table_name</i>, netobj *<i>cookie</i>);  void <b>nis_freeresult</b>(nis_result *<i>result</i>); </pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>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:</p> <pre>[ colname=value, . . . ],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> 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.</p> <p><b>FOLLOW_LINKS</b> If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3NSL)</code>), 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 <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

## nis\_tables(3NSL)

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 <i>name</i> 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 <i>only</i> 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 <code>nis_getnames()</code> [see <code>nis_local_names(3NSL)</code> ] which uses the environment variable <code>NIS_PATH</code> .
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the <code>nis_result</code> 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.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`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 `LEN_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

`nis_tables(3NSL)`

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 {
        uint_t    objects_len;
        nis_object *objects_val;
    } objects;
    netobj       cookie;
    uint32_t      zticks;
    uint32_t      dticks;
    uint32_t      aticks;
    uint32_t      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(3NSL)`].

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(3NSL)`). 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(3NSL)`).

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.



<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 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.
<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.

## 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 *expired*. 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

`nis_tables(3NSL)`

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 *object* 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(3C)` 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(3C)` 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

nis\_tables(3NSL)

## ENVIRONMENT VARIABLES

## ATTRIBUTES

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

SunOS 5.8  
Reference Manual

## WARNINGS

## NOTES

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.

**NIS\_PATH** When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3NSL)`, `nis_server(3NSL)`,  
`rpc_svc_calls(3NSL)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3C)`, `nis_clone_object(3NSL)`,  
`nis_destroy_object(3NSL)`, `nis_error(3NSL)`, `nis_getnames(3NSL)`,  
`nis_local_names(3NSL)`, `nis_objects(3NSL)`, `attributes(5)`

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

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.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3NSL)`. Otherwise, this function is MT-Safe.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpcsvc/nis.h&gt;  bool_t <b>nis_ismember</b>(nis_name <i>principal</i>, nis_name <i>group</i>) ; nis_error <b>nis_addmember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_removemember</b>(nis_name <i>member</i>, nis_name <i>group</i>) ; nis_error <b>nis_creategroup</b>(nis_name <i>group</i>, uint_t <i>flags</i>) ; nis_error <b>nis_destroygroup</b>(nis_name <i>group</i>) ; void <b>nis_print_group_entry</b>(nis_name <i>group</i>) ; nis_error <b>nis_verifygroup</b>(nis_name <i>group</i>) ;</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>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.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> <li>■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> <li>■ 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.</li> <li>■ 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.</li> </ul> <p>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.</p> <p>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.</p> <p>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.</p> <p>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</p>

## `nis_verifygroup(3NSL)`

and modify rights for the group in question. To succeed, `nis_addmember()` 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

#### EXAMPLE 1 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).
```

#### EXAMPLE 2 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).
```

#### EXAMPLE 3 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.
```

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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

`nisgrpadm(1)`, `nis_objects(3NSL)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

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

```
uint_t    gr_flags;      /* Interpretation Flags
                          (currently unused) */
struct {
    uint_t  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.

## plock(3C)

<b>NAME</b>	plock – Lock or unlock into memory process, text, or data								
<b>SYNOPSIS</b>	<pre>#include &lt;sys/lock.h&gt;  int <b>plock</b>(int <i>op</i>);</pre>								
<b>DESCRIPTION</b>	<p>The <code>plock()</code> function 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 calling process must have the <code>PRIV_SYS_CONFIG</code> privilege to succeed.</p> <p>The <code>plock()</code> function performs the function specified by <i>op</i>:</p> <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.								
<b>RETURN VALUES</b>	<p><code>plock()</code> returns:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.				
0	On success.								
-1	On failure, and sets <code>errno</code> to indicate the error.								
<b>ERRORS</b>	<p>The <code>plock()</code> function fails and does not perform the requested operation if:</p> <table> <tr> <td>EAGAIN</td><td>Not enough memory.</td></tr> <tr> <td>EINVAL</td><td>The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> 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	The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> and no lock exists on the calling process.	EPERM	The process does not have sufficient privilege.		
EAGAIN	Not enough memory.								
EINVAL	The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> and no lock exists on the calling process.								
EPERM	The process does not have sufficient privilege.								
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>To succeed, <code>plock()</code> must have <code>PRIV_SYS_CONFIG</code> in its set of effective privileges.</p> <p>The <code>mlock(3C)</code> and <code>mlockall(3C)</code> functions are the preferred interfaces for process locking.</p>								
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>exec(2)</code>, <code>fork(2)</code>, <code>mlock(3C)</code>, <code>mlockall(3C)</code></p> <p><code>exit(2)</code>, <code>memcntl(2)</code></p>								



NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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.</p> <p>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 <code>file_mac_read,file_mac_write,none,proc_nofloat</code> produces the same result as <code>proc_nofloat</code> alone. The constructed privilege set is stored in the <i>priv_set_t</i> buffer to which <i>priv_set</i> points.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

priv\_set\_to\_str(3TSOL)

RETURN VALUES	get_priv_text()	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
	priv_to_str()	Returns a pointer to the translated privilege name string. The function returns NULL and sets <i>errno</i> on failure.
	str_to_priv()	Returns the numeric privilege ID. The function returns -1 and sets <i>errno</i> 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 NULL. The function returns NULL and sets <i>errno</i> on failure. This function returns -1 if the string cannot be translated or if an integer ordinal in the string is greater than TSOL_MAX_PRIV.
	str_to_priv_set()	Returns NULL 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.
ERRORS	priv_to_str() may fail for this reason:	
	EINVAL	The specified <i>priv_id</i> is greater than TSOL_MAX_PRIV.
	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.
NOTES	EFAULT	The specified <i>priv_name</i> is an invalid address.
	To use these routines, the program must be loaded with the Trusted Solaris library libtsol or libtsol.so.	
Trusted Solaris 8 4/01 Reference Manual	priv_desc(4)	priv_name(4)
	attributes(5)	

NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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.</p> <p>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 <code>file_mac_read,file_mac_write,none,proc_nofloat</code> produces the same result as <code>proc_nofloat</code> alone. The constructed privilege set is stored in the <i>priv_set_t</i> buffer to which <i>priv_set</i> points.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

priv\_to\_str(3TSOL)

**RETURN VALUES**

get_priv_text()	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
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 NULL. The function returns NULL 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 TSOL_MAX_PRIV.
str_to_priv_set()	Returns NULL 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.

**ERRORS**

priv_to_str() may fail for this reason:	
EINVAL	The specified <i>priv_id</i> is greater than TSOL_MAX_PRIV.
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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.10  
Reference Manual

priv\_desc(4) priv\_name(4)  
attributes(5)

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] ( <b>obsolete</b> )
DESCRIPTION	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

## pututline(3C)

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];    /* user login name */ char          ut_id[4];     /* /sbin/inittab id (usually line #) */ char          ut_line[12];  /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;           /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

pututline(3C)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

**SunOS 5.8  
Reference Manual  
NOTES**

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.

ttyslot(3C), utmp(4), utmpx(4), attributes(5)

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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().



NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database 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</p>

## pututxline(3C)

	<p><i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i>⇒<code>ut_id</code>. If the end of database is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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>⇒<code>ut_line</code> string. If the end of the database is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open database.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)</p>

pututxline(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

## pututxline(3C)

The `endutxent()` and `setutxent()` functions return no value.

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

### USAGE

These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

### FILES

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

### 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

### Trusted Solaris 8 4/01 Reference Manual NOTES

`getutent(3C)`

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

randomword(3TSOL)

NAME	randomword – Generate random pronounceable password						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library ]  #include &lt;tsol/tsol.h&gt;  int randomword(char *word, char *hyphenated_word, const unsigned     short minlen, const unsigned short maxlen, const unsigned char     *seed);</pre>						
DESCRIPTION	<p>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>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Unsafe						
RETURN VALUES	<p>randomword() returns:</p> <ul style="list-style-type: none"><li>-1 If <i>minlen</i> &gt; <i>maxlen</i> or <i>seed</i> has not been set.</li><li>0 If <i>maxlen</i> is zero (0).</li><li>&gt;0 Length of the password <i>word</i> generated.</li></ul>						
EXAMPLES	<p><b>EXAMPLE 1</b> Randomword Example</p> <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 &lt; 5; i++) {     len = randomword(password, hyphen_password, 6, 8, (unsigned char *) 0);     printf("password %s is pronounced %s\n", password, hyphen_password); }</pre>						

randomword(3TSOL)

**Trusted Solaris 8  
4/01 Reference  
Manual**

**SunOS 5.8  
Reference Manual**

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

attributes(5)

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_hostalias(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
               char *data, int datalen, struct rrec *newrr, u_char *buf, int
               buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
             anslen);

void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with RES_USEVC to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.



	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_hostalias(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

	res_hostalias(3RESOLV)
<b>res_hostalias</b>	<p>The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i>.</p>
<b>res_nclose</b>	<p>The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i>.</p>
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	<p>The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i>, which is of size <i>length</i>. <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<p>/etc/resolv.conf                      Resolver configuration file</p>
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p>

res\_hostalias(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_init(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

res\_init(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_init(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>



res\_init(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_init(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_mkquery(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_mkquery(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_sendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_sendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_sendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

	res_mkquery(3RESOLV)
<b>res_hostalias</b>	<p>The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i>.</p>
<b>res_nclose</b>	<p>The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i>.</p>
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	<p>The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i>, which is of size <i>length</i>. <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<p>/etc/resolv.conf                      Resolver configuration file</p>
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p>

res\_mkquery(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.



NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_nclose(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with RES_USEVC to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_nclose(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_sendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_sendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_sendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

res\_nclose(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nclose(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>– resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_ninit(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.



	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_ninit(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

res\_ninit(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_ninit(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herron – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_nmkquery(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with RES_USEVC to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_nmkquery(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>



res\_nmkquery(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nmkquery(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`

`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_npquery(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_npquery(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

	res_npquery(3RESOLV)
<b>res_hostalias</b>	<p>The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i>.</p>
<b>res_nclose</b>	<p>The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i>.</p>
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	<p>The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i>, which is of size <i>length</i>. <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<p>/etc/resolv.conf                      Resolver configuration file</p>
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p>

res\_npquery(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.



NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_nquery(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_nquery(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

res\_nquery(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nquery(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_nquerydomain(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.



## res\_nquerydomain(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_nquerydomain(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmkquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

	res_nquerydomain(3RESOLV)
<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	/etc/resolv.conf                      Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nquerydomain(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>– resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket      -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## res\_nsearch(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
              int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
               int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
                char *data, int datalen, struct rrec *newrr, u_char *buf, int  
                buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
             anslen);  
void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions `res_init()`, `res_query()`, `res_search()`, `res_mkquery()`, `res_send()`, and `herror()` are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure `_res` rather than state information referenced through *statp*.

Most of the values in *statp* and `_res` are initialized to reasonable defaults on the first call to `res_ninit()` or `res_init()` and can be ignored. Options stored in `statp->options` or `_res.options` are defined in `<resolv.h>`. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> or <code>res_ninit()</code> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <code>RES_USEVC</code> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_nsearch(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_sendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_sendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_sendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>



<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <code>statp-&gt;res_h_errno</code> and <code>_res.res_h_errno</code> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nsearch(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_nsend(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_nsend(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

res\_nsend(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_nsend(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.



NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket      -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_nsendsigned(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

res\_nsendsigned(3RESOLV)

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_nsendsigned(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

res\_nsendsigned(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See <code>attributes(5)</code> for descriptions of the following attributes:

res\_nsendsigned(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

## resolver(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

### DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.



	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## resolver(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <code>res_nquery()</code> and <code>res_query()</code> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <code>res_nquery()</code> and <code>res_query()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <code>res_nsearch()</code> and <code>res_search()</code> routines make a query and await a response, just like like <code>res_nquery()</code> and <code>res_query()</code>. In addition, they implement the default and search rules controlled by the <code>RES_DEFNAMES</code> and <code>RES_DNSRCH</code> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <code>res_nsearch()</code> and <code>res_search()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <code>res_nquery()</code> and <code>res_query()</code>. The <code>res_nmquery()</code> and <code>res_mkquery()</code> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <code>QUERY</code>, but can be any of the query types defined in <code>&lt;arpa/nameser.h&gt;</code>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <code>res_nsend()</code>, <code>res_send()</code>, and <code>res_nsendsigned()</code> routines send a preformatted query that returns an <i>answer</i>. The routine calls <code>res_ninit()</code> or <code>res_init()</code>. If <code>RES_INIT</code> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <code>res_nsendsigned()</code> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <code>res_nsend()</code> and <code>res_send()</code> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <code>res_npquery()</code> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <code>fp_resstat()</code> function prints out the active flag bits in <code>statp-&gt;options</code> preceded by the text <code>"; ; res options:"</code> on <i>file</i>.</p>

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

resolver(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herror – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_query(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

## res\_query(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>



res\_query(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_query(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herrordata-cs="2" data-kind="parent">>– resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_search(3RESOLV)

```
int res_init(void);  
int res_query(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_search(const char *dname, int class, int type, u_char *answer,  
int anslen);  
int res_mkquery(int op, const char *dname, int class, int type, const  
char *data, int datalen, struct rrec *newrr, u_char *buf, int  
buflen);  
int res_send(const u_char *msg, int msglen, u_char *answer, int  
anslen);  
void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the search command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_search(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmkquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmkquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <code>statp-&gt;res_h_errno</code> and <code>_res.res_h_errno</code> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_search(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

**NOTES**

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.



NAME	resolver, res_ninit, fp_resstat, res_npquery, res_hostalias, res_nquery, res_nsearch, res_nquerydomain, res_nmkquery, res_nsend, res_nclose, res_nsendsigned, dn_comp, dn_expand, hstrerror, res_init, res_query, res_search, res_mkquery, res_send, herron – resolver routines
BIND 8.2.2 Interfaces	<pre>cc [ flag ... ] file ... -lresolv  -lsocket  -lnsl  [ library ... ] #include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;  int res_ninit(res_state statp);  void fp_resstat(const res_state statp, FILE *fp);  void res_npquery(const res_state statp, const u_char *msg, int     msglen, FILE *fp);  const char *res_hostalias(const res_state statp, const char *name,     char * name, char *buf, size_tbuflen);  int res_nquery(res_state statp, const char *dname, int class, int type,     u_char *answer, int datalen, int anslen);  int res_nsearch(res_state statp, const char *dname, int class, int     type, u_char *answer, int anslen);  int res_nquerydomain(res_state statp, const char *name, const char     *domain, int class, int type, u_char *answer, int anslen);  int res_nmkquery(res_state statp, int op, const char *dname, int     class, int type, u_char *answer, int datalen, int anslen);  int res_nsend(res_state statp, const u_char *msg, int msglen, u_char     *answer, int anslen);  void res_nclose(res_state statp);  int res_nsendsigned(res_state statp, const u_char *msg, int msglen,     ns_tsig_key *key, u_char *answer, int anslen);  int dn_comp(const char *exp_dn, u_char *comp_dn, int length, u_char     **dnptrs, **lastdnptr);  int dn_expand(const u_char *msg, *eomorig, *comp_dn, char *exp_dn,     int length);  const char *hstrerror(int err);</pre>
Deprecated Interfaces	<pre>#include &lt;sys/types.h&gt; #include &lt;netinet/in.h&gt; #include &lt;arpa/nameser.h&gt; #include &lt;resolv.h&gt; #include &lt;netdb.h&gt;</pre>

res\_send(3RESOLV)

```
int res_init(void);

int res_query(const char *dname, int class, int type, u_char *answer,
              int anslen);

int res_search(const char *dname, int class, int type, u_char *answer,
               int anslen);

int res_mkquery(int op, const char *dname, int class, int type, const
                char *data, int datalen, struct rrec *newrr, u_char *buf, int
                buflen);

int res_send(const u_char *msg, int msglen, u_char *answer, int
              anslen);

void herror(const char *s);
```

## DESCRIPTION

These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.

State information is kept in *statp* and is used to control the behavior of these functions. Set *statp* to all zeros prior to making the first call to any of these functions.

The functions *res\_init()*, *res\_query()*, *res\_search()*, *res\_mkquery()*, *res\_send()*, and *herror()* are deprecated. They are supplied for backwards compatibility. They use global configuration and state information that is kept in the structure *\_res* rather than state information referenced through *statp*.

Most of the values in *statp* and *\_res* are initialized to reasonable defaults on the first call to *res\_ninit()* or *res\_init()* and can be ignored. Options stored in *statp->options* or *\_res.options* are defined in *<resolv.h>*. They are stored as a simple bit mask containing the bitwise OR of the options enabled.

RES_INIT	True if the initial name server address and default domain name are initialized, that is, <i>res_init()</i> or <i>res_ninit()</i> has been called.
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. With this option, <i>res_send()</i> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
RES_USEVC	Use TCP connections for queries instead of UDP datagrams.
RES_STAYOPEN	Use with <i>RES_USEVC</i> to keep the TCP connection open between queries. This is a useful option for programs that regularly do many queries. The normal mode used should be UDP.
RES_IGNTC	Ignore truncation errors; that is, do not retry with TCP.

	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> and <code>res_nsend()</code> do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, <code>res_search()</code> and <code>res_nsearch()</code> append the default domain name to single-component names, that is, names that do not contain a dot. This option is enabled by default.
	RES_DNSRCH	If this option is set, <code>res_search()</code> and <code>res_nsearch()</code> search for host names in the current domain and in parent domains. See <code>hostname(1)</code> . This option is used by the standard host lookup routine <code>gethostbyname(3NSL)</code> . This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.
	RES_ROTATE	This option causes <code>res_nsend()</code> and <code>res_send()</code> to rotate the list of nameservers in <code>statp-&gt;nsaddr_list</code> or <code>_res.nsaddr_list</code> .
	RES_KEEPTSIG	This option causes <code>res_nsendsigned()</code> to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
<b>res_ninit, res_init</b>		The <code>res_ninit()</code> and <code>res_init()</code> routines read the configuration file, if any is present, to get the default domain name, search list and the Internet address of the local name server(s). See <code>resolv.conf(4)</code> . If no server is configured, <code>res_init()</code> or <code>res_ninit()</code> will try to obtain name resolution services from the host on which it is running. The current domain name is defined by <code>domainname(1M)</code> , or by the <code>hostname</code> if it is not specified in the configuration file. Use the environment variable <code>LOCALDOMAIN</code> to override the domain name. This environment variable may contain several blank-separated tokens if you wish to override the search list on a per-process basis. This is similar to the <code>search</code> command in the configuration file. You can set the <code>RES_OPTIONS</code> environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the <code>statp / _res</code> structure. Alternatively, they are inherited from the configuration file's <code>options</code> command. See <code>resolv.conf(4)</code> for information regarding the syntax of the <code>RES_OPTIONS</code> environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
<b>res_nquery, res_query</b>		The <code>res_nquery()</code> and <code>res_query()</code> functions provide interfaces to the server query mechanism. They construct a query, send it to the local server, await a response, and make preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i> . The reply

res\_send(3RESOLV)

	<p>message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. <i>res_nquery()</i> and <i>res_query()</i> return the length of the <i>answer</i>, or -1 upon error.</p> <p>The <i>res_nquery()</i> and <i>res_query()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nsearch, res_search</b>	<p>The <i>res_nsearch()</i> and <i>res_search()</i> routines make a query and await a response, just like like <i>res_nquery()</i> and <i>res_query()</i>. In addition, they implement the default and search rules controlled by the <i>RES_DEFNAMES</i> and <i>RES_DNSRCH</i> options. They return the length of the first successful reply which is stored in <i>answer</i>. On error, they return -1.</p> <p>The <i>res_nsearch()</i> and <i>res_search()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_nmquery, res_mkquery</b>	<p>These routines are used by <i>res_nquery()</i> and <i>res_query()</i>. The <i>res_nmquery()</i> and <i>res_mkquery()</i> functions construct a standard query message and place it in <i>buf</i>. The routine returns the <i>size</i> of the query, or -1 if the query is larger than <i>buflen</i>. The query type <i>op</i> is usually <i>QUERY</i>, but can be any of the query types defined in <i>&lt;arpa/nameser.h&gt;</i>. The domain name for the query is given by <i>dname</i>. <i>newrr</i> is currently unused but is intended for making update messages.</p>
<b>res_nsend, res_send, res_nsendsigned</b>	<p>The <i>res_nsend()</i>, <i>res_send()</i>, and <i>res_nsendsigned()</i> routines send a preformatted query that returns an <i>answer</i>. The routine calls <i>res_ninit()</i> or <i>res_init()</i>. If <i>RES_INIT</i> is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the <i>res_nsendsigned()</i> uses TSIG signatures to add authentication to the query and verify the response. In this case, only one name server will be contacted. The routines return the length of the reply message, or -1 if there are errors.</p> <p>The <i>res_nsend()</i> and <i>res_send()</i> routines return a length that may be bigger than <i>anslen</i>. In that case, retry the query with a larger <i>buf</i>. The <i>answer</i> to the second query may be larger still], so it is recommended that you supply a <i>buf</i> larger than the <i>answer</i> returned by the previous query. <i>answer</i> must be large enough to receive a maximum UDP response from the server or parts of the <i>answer</i> will be silently discarded. The default maximum UDP response size is 512 bytes.</p>
<b>res_npquery</b>	<p>The <i>res_npquery()</i> function prints out the query and any answer in <i>msg</i> on <i>fp</i>.</p>
<b>fp_resstat</b>	<p>The <i>fp_resstat()</i> function prints out the active flag bits in <i>statp-&gt;options</i> preceded by the text <i>";; res options:"</i> on <i>file</i>.</p>

res\_send(3RESOLV)

<b>res_hostalias</b>	The <code>res_hostalias()</code> function looks up <i>name</i> in the file referred to by the HOSTALIASES environment variable and returns the fully qualified host name. If <i>name</i> is not found or an error occurs, NULL is returned. <code>res_hostalias()</code> stores the result in <i>buf</i> .
<b>res_nclose</b>	The <code>res_nclose()</code> function closes any open files referenced through <i>statp</i> .
<b>dn_comp</b>	<p>The <code>dn_comp()</code> function compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>.</p> <p><i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, <code>dn_comp()</code> does not update the list of labels.</p>
<b>dn_expand</b>	The <code>dn_expand()</code> function expands the compressed domain name <i>comp_dn</i> to a full domain name. The compressed name is contained in a query or reply message. <i>msg</i> is a pointer to the beginning of that message. The uncompressed name is placed in the buffer indicated by <i>exp_dn</i> , which is of size <i>length</i> . <code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.
<b>hstrerror, herror</b>	<p>The variables <i>statp-&gt;res_h_errno</i> and <i>_res.res_h_errno</i> and external variable <i>h_errno</i> are set whenever an error occurs during a resolver operation. The following definitions are given in <code>&lt;netdb.h&gt;</code>:</p> <pre>#define NETDB_INTERNAL -1 /* see errno */ #define NETDB_SUCCESS 0 /* no problem */ #define HOST_NOT_FOUND 1 /* Authoritative Answer Host not found */ #define TRY_AGAIN 2 /* Non-Authoritative not found, or SERVFAIL */ #define NO_RECOVERY 3 /* Non-Recoverable: FORMERR, REFUSED, NOTIMP */ #define NO_DATA 4 /* Valid name, no data for requested type */</pre> <p>The <code>herror()</code> function writes a message to the diagnostic output consisting of the string parameters, the constant string ":", and a message corresponding to the value of <i>h_errno</i>.</p> <p>The <code>hstrerror()</code> function returns a string, which is the message text that corresponds to the value of the <i>err</i> parameter.</p>
<b>FILES</b>	<code>/etc/resolv.conf</code> Resolver configuration file
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

res\_send(3RESOLV)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit) SUNWcslx (64-bit)
Interface Stability	Standard BIND 8.2.2
MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.

#### SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, then the functions `res_nsend()`, `res_send()`, `res_nsearch()`, 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 the functions `res_nsend()`, `res_send()`, `res_nsearch()`, and `res_search()` can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.

#### Trusted Solaris 8 4/01 Reference Manual

`in.named(1M)`, `resolv.conf(4)`  
`domainname(1M)`, `gethostbyname(3NSL)`, `libresolv(3LIB)`, `attributes(5)`  
Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.  
Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.  
Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND (public domain)*, Internet Software Consortium, 1996.

#### NOTES

When the caller supplies a work buffer, for example the *answer* buffer argument to `res_nsend()` or `res_send()`, the buffer should be aligned on an eight byte boundary. Otherwise, an error such as a `SIGBUS` may result.

<b>NAME</b>	rpc – Library routines for remote procedure calls						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lnsl [library...]  #include &lt;rpc/rpc.h&gt; #include &lt;netconfig.h&gt;</pre>						
<b>DESCRIPTION</b>	<p>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.</p> <p>All RPC routines require the header <code>&lt;rpc/rpc.h&gt;</code>. Routines that take a <code>netconfig</code> structure also require that <code>&lt;netconfig.h&gt;</code> be included. Applications using RPC and XDR routines should be linked with the <code>libnsl</code> library.</p>						
<b>Multithread Considerations</b>	<p>In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). Defining this flag enables a thread-specific version of <code>rpc_clnt_create</code>(3NSL). See <code>rpc_clnt_create</code>(3NSL).</p> <p>When used in multithreaded applications, client-side routines are MT-Safe. 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). See <code>rpc_clnt_create</code>(3NSL).</p> <p>When used in multithreaded applications, server-side routines are usually Unsafe. In this implementation the service transport handle, <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. See <code>rpc_svc_create</code>(3NSL). 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 <code>rpc_svc_calls</code>(3NSL).</p>						
<b>Nettyp</b>	<p>Some of the high-level RPC interface routines take a <i>nettype</i> string as one of the parameters (for example, <code>clnt_create()</code>, <code>svc_create()</code>, <code>rpc_reg()</code>, <code>rpc_call()</code>). This string defines a class of transports which can be used for a particular application.</p> <p><i>nettype</i> can be one of the following:</p> <table> <tr> <td><code>netpath</code></td><td>Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or <code>NULL</code>, it defaults to <code>visible</code>. <code>netpath</code> is the default <i>nettype</i>.</td></tr> <tr> <td><code>visible</code></td><td>Choose the transports which have the visible flag (<code>v</code>) set in the <code>/etc/netconfig</code> file.</td></tr> <tr> <td><code>circuit_v</code></td><td>This is same as <code>visible</code> except that it chooses only the connection oriented transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code>) from the entries in the <code>/etc/netconfig</code> file.</td></tr> </table>	<code>netpath</code>	Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or <code>NULL</code> , it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> .	<code>visible</code>	Choose the transports which have the visible flag ( <code>v</code> ) set in the <code>/etc/netconfig</code> file.	<code>circuit_v</code>	This is same as <code>visible</code> except that it chooses only the connection oriented transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code> ) from the entries in the <code>/etc/netconfig</code> file.
<code>netpath</code>	Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or <code>NULL</code> , it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> .						
<code>visible</code>	Choose the transports which have the visible flag ( <code>v</code> ) set in the <code>/etc/netconfig</code> file.						
<code>circuit_v</code>	This is same as <code>visible</code> except that it chooses only the connection oriented transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code> ) from the entries in the <code>/etc/netconfig</code> file.						

rpc(3NSL)

	<code>datagram_v</code>	This is same as <code>visible</code> except that it chooses only the connectionless datagram transports (semantics <code>tpi_clts</code> ) from the entries in the <code>/etc/netconfig</code> file.
	<code>circuit_n</code>	This is same as <code>netpath</code> except that it chooses only the connection oriented datagram transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code> ).
	<code>datagram_n</code>	This is same as <code>netpath</code> except that it chooses only the connectionless datagram transports (semantics <code>tpi_clts</code> ).
	<code>udp</code>	This refers to Internet UDP.
	<code>tcp</code>	This refers to Internet TCP.
	If <i>nettype</i> is NULL, it defaults to <code>netpath</code> . The transports are tried in left to right order in the <code>NETPATH</code> variable or in top to down order in the <code>/etc/netconfig</code> file.	
Derived Types	In a 64-bit environment, the derived types are defined as follows:	
	<code>typedef</code>	<code>uint32_t</code> <code>rpcprog_t;</code>
	<code>typedef</code>	<code>uint32_t</code> <code>rpcvers_t;</code>
	<code>typedef</code>	<code>uint32_t</code> <code>rpcproc_t;</code>
	<code>typedef</code>	<code>uint32_t</code> <code>rpcprot_t;</code>
	<code>typedef</code>	<code>uint32_t</code> <code>rpcport_t;</code>
	<code>typedef</code>	<code>int32_t</code> <code>rpc_inline_t;</code>
	In a 32-bit environment, the derived types are defined as follows:	
	<code>typedef</code>	<code>unsigned long</code> <code>rpcprog_t;</code>
	<code>typedef</code>	<code>unsigned long</code> <code>rpcvers_t;</code>
	<code>typedef</code>	<code>unsigned long</code> <code>rpcproc_t;</code>
	<code>typedef</code>	<code>unsigned long</code> <code>rpcprot_t;</code>
	<code>typedef</code>	<code>unsigned long</code> <code>rpcport_t;</code>
	<code>typedef</code>	<code>long</code> <code>rpc_inline_t;</code>
Data Structures	Some of the data structures used by the RPC package are shown below.	
The AUTH Structure	<pre>union des_block {     struct {         u_int32 high;         u_int32 low;     }; };</pre>	



**The CLIENT  
Structure**

```

        } 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 */
    uint_t 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 */
        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 */
        int (*cl_settimers)( );    /* set rpc level timers */
    } *cl_ops;
    caddr_t cl_private;    /* private stuff */
    char *cl_netid;        /* network identifier */
    char *cl_tp;           /* device name */
} CLIENT;

enum xprt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};
/*
 * Server side transport handle

```

**The SVCXPRT  
Structure**

## rpc(3NSL)

**The svc\_req  
Structure**

**The XDR  
Structure**

```

*/
typedef struct {
    int xp_fd; /* file descriptor for the
    ushort_t xp_port; /* obsolete */
    struct xp_ops {
        bool_t (*xp_rcv)(); /* receive incoming requests */
        enum xpstat (*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 */
} SVCXPRT;

struct svc_req {
    rpcprog_t rq_prog; /* service program number */
    rpcvers_t rq_vers; /* service protocol version */
    rpcproc_t 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 */
};

/*
 * 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
 * an 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 long from underlying stream */
bool_t (*x_putlong)(); /* put long to underlying stream */
bool_t (*x_getbytes)(); /* get bytes from underlying stream */
bool_t (*x_putbytes)(); /* put bytes to underlying stream */
uint_t (*x_getpostn)(); /* returns bytes off from beginning */
bool_t (*x_setpostn)(); /* reposition the stream */
rpc_inline_t (*x_inline)(); /* buf quick ptr to buffered data */
void (*x_destroy)(); /* free privates of this xdr_stream */
bool_t (*x_control)(); /* changed/retrieve client object info */
bool_t (*x_getint32)(); /* get int from underlying stream */
bool_t (*x_putint32)(); /* put int to underlying stream */

} *x_ops;

caddr_t x_public; /* users' data */
caddr_t x_priv; /* 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 index lists RPC routines and the manual reference pages on which they are described:

RPC Routine	Manual Reference Page
auth_destroy()	rpc_clnt_auth(3NSL)
authdes_create()	rpc_soc(3NSL)
authdes_getucred()	secure_rpc(3NSL)
authdes_seccreate()	secure_rpc(3NSL)
authkerb_getucred()	kerberos_rpc(3KRB)
authkerb_seccreate()	kerberos_rpc(3KRB)

## rpc(3NSL)

authnone_create()	rpc_clnt_auth(3NSL)
authsys_create()	rpc_clnt_auth(3NSL)
authsys_create_default()	rpc_clnt_auth(3NSL)
authunix_create()	rpc_soc(3NSL)
authunix_create_default()	rpc_soc(3NSL)
callrpc()	rpc_soc(3NSL)
clnt_broadcast()	rpc_soc(3NSL)
clnt_call()	rpc_clnt_calls(3NSL)
clnt_control()	rpc_clnt_create(3NSL)
clnt_create()	rpc_clnt_create(3NSL)
clnt_destroy()	rpc_clnt_create(3NSL)
clnt_dg_create()	rpc_clnt_create(3NSL)
clnt_freeres()	rpc_clnt_calls(3NSL)
clnt_geterr()	rpc_clnt_calls(3NSL)
clnt_pcreateerror()	rpc_clnt_create(3NSL)
clnt_perrno()	rpc_clnt_calls(3NSL)
clnt_perror()	rpc_clnt_calls(3NSL)
clnt_raw_create()	rpc_clnt_create(3NSL)
clnt_spccreateerror()	rpc_clnt_create(3NSL)
clnt_sperrno()	rpc_clnt_calls(3NSL)
clnt_sperror()	rpc_clnt_calls(3NSL)
clnt_tli_create()	rpc_clnt_create(3NSL)
clnt_tp_create()	rpc_clnt_create(3NSL)
clnt_udpcreate()	rpc_soc(3NSL)
clnt_vc_create()	rpc_clnt_create(3NSL)
clntraw_create()	rpc_soc(3NSL)
clnttcp_create()	rpc_soc(3NSL)
clntudp_bufcreate()	rpc_soc(3NSL)
get_myaddress()	rpc_soc(3NSL)
getnetname()	secure_rpc(3NSL)
host2netname()	secure_rpc(3NSL)

rpc(3NSL)

key_decryptsession()	secure_rpc(3NSL)
key_encryptsession()	secure_rpc(3NSL)
key_gendes()	secure_rpc(3NSL)
key_setsecret()	secure_rpc(3NSL)
netname2host()	
netname2user()	secure_rpc(3NSL)
pmap_getmaps()	rpc_soc(3NSL)
pmap_getport()	rpc_soc(3NSL)
pmap_rmtcall()	rpc_soc(3NSL)
pmap_set()	rpc_soc(3NSL)
pmap_unset()	rpc_soc(3NSL)
rac_drop()	rpc_rac(3RAC)
rac_poll()	rpc_rac(3RAC)
rac_recv()	rpc_rac(3RAC)
rac_send()	rpc_rac(3RAC)
registerrpc()	rpc_soc(3NSL)
rpc_broadcast()	rpc_clnt_calls(3NSL)
rpc_broadcast_exp()	rpc_clnt_calls(3NSL)
rpc_call()	rpc_clnt_calls(3NSL)
rpc_reg()	rpc_svc_calls(3NSL)
svc_create()	rpc_svc_create(3NSL)
svc_destroy()	rpc_svc_create(3NSL)
svc_dg_create()	rpc_svc_create(3NSL)
svc_dg_enablecache()	rpc_svc_calls(3NSL)
svc_fd_create()	rpc_svc_create(3NSL)
svc_fds()	rpc_soc(3NSL)
svc_freeargs()	rpc_svc_reg(3NSL)
svc_getargs()	rpc_svc_reg(3NSL)
svc_getcaller()	rpc_soc(3NSL)
svc_getreq()	rpc_soc(3NSL)
svc_getreqset()	rpc_svc_calls(3NSL)

## rpc(3NSL)

<code>svc_getrpccaller()</code>	<code>rpc_svc_calls(3NSL)</code>
<code>svc_kerb_reg()</code>	<code>kerberos_rpc(3KRB)</code>
<code>svc_raw_create()</code>	<code>rpc_svc_create(3NSL)</code>
<code>svc_reg()</code>	<code>rpc_svc_calls(3NSL)</code>
<code>svc_register()</code>	<code>rpc_soc(3NSL)</code>
<code>svc_run()</code>	<code>rpc_svc_reg(3NSL)</code>
<code>svc_sendreply()</code>	<code>rpc_svc_reg(3NSL)</code>
<code>svc_tli_create()</code>	<code>rpc_svc_create(3NSL)</code>
<code>svc_tp_create()</code>	<code>rpc_svc_create(3NSL)</code>
<code>svc_unreg()</code>	<code>rpc_svc_calls(3NSL)</code>
<code>svc_unregister()</code>	<code>rpc_soc(3NSL)</code>
<code>svc_vc_create()</code>	<code>rpc_svc_create(3NSL)</code>
<code>svcerr_auth()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_decode()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_noproc()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_noprog()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_progvers()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_systemerr()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcerr_weakauth()</code>	<code>rpc_svc_err(3NSL)</code>
<code>svcfld_create()</code>	<code>rpc_soc(3NSL)</code>
<code>svccraw_create()</code>	<code>rpc_soc(3NSL)</code>
<code>svctcp_create()</code>	<code>rpc_soc(3NSL)</code>
<code>svcudp_bufcreate()</code>	<code>rpc_soc(3NSL)</code>
<code>svcudp_create()</code>	<code>rpc_soc(3NSL)</code>
<code>user2netname()</code>	<code>secure_rpc(3NSL)</code>
<code>xdr_accepted_reply()</code>	<code>rpc_xdr(3NSL)</code>
<code>xdr_authsys_parms()</code>	<code>rpc_xdr(3NSL)</code>
<code>xdr_authunix_parms()</code>	<code>rpc_soc(3NSL)</code>
<code>xdr_callhdr()</code>	<code>rpc_xdr(3NSL)</code>
<code>xdr_callmsg()</code>	<code>rpc_xdr(3NSL)</code>
<code>xdr_opaque_auth()</code>	<code>rpc_xdr(3NSL)</code>

xdr_rejected_reply()	rpc_xdr(3NSL)
xdr_replymsg()	rpc_xdr(3NSL)
xprt_register()	rpc_svc_calls(3NSL)
xprt_unregister()	rpc_svc_calls(3NSL)

**FILES** /etc/netconfig Network configuration database

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**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.

**Trusted Solaris 8 4/01 Reference Manual** libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL), rpc\_clnt\_calls(3NSL), rpc\_clnt\_create(3NSL), rpc\_svc\_calls(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), rpcbind(3NSL)

**SunOS 5.8 Reference Manual** getnetconfig(3NSL), getnetpath(3NSL), kerberos\_rpc(3KRB), rpc\_clnt\_auth(3NSL), rpc\_svc\_err(3NSL), rpc\_xdr(3NSL), secure\_rpc(3NSL), xdr(3N), netconfig(4), rpc(4), attributes(5), environ(5)

## rpcb\_getaddr(3NSL)

<b>NAME</b>	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
<b>SYNOPSIS</b>	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
<b>DESCRIPTION</b>	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
<b>Routines</b>	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()</pre> <p>An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL if the remote rpcbind could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>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.</p>



### rpcb\_getaddr()

An interface to the rpcbnd service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote rpcbnd service. In the last case, the global variable *rpc\_createerr* contains the RPC status. See *rpc\_clnt\_create(3NSL)*.

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.

### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, *rpcb\_gettime()* returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

### rpcb\_rmtcall()

An interface to the rpcbnd service, which instructs rpcbnd 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(3NSL)* 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 rpcbnd does not return any error messages to the caller. In such a case, the caller times out.

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

Note: *rpcb\_rmtcall()* is only available for connectionless transports.

### rpcb\_set()

An interface to the rpcbnd service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf*⇒*nc\_netid*] and *svcaddr* on the machine's rpcbnd 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.

## rpcb\_getaddr(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

### rpcb\_unset()

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`

`attributes(5)`

NAME	<p>rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service</p>
SYNOPSIS	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *rpcb_getmaps(const struct netconfig *netconf,                              const char *host);  struct tsol_rpcblist *rpcb_getallmaps(const struct netconfig                                       *netconf, const char *host);  bool_t rpcb_getaddr(const rpcprog_t prognum, const rpcvers_t                    versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,                    const char *host);  bool_t rpcb_gettime(const char *host, time_t *timep);  enum clnt_stat rpcb_rmtcall(const struct netconfig *netconf, const                            char *host, const rpcprog_t prognum, const rpcvers_t versnum,                            const rpcproc_t proct, const xdrproc_t inproc, const caddr_t                            in, const xdrproc_t outproc, caddr_t out, const struct timeval                            tout, struct netbuf *svcaddr);  bool_t rpcb_set(const rpcprog_t prognum, const rpcvers_t versnum,                const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t rpcb_unset(const rpcprog_t prognum, const rpcvers_t versnum,                  const struct netconfig *netconf);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
Routines	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()     An interface to the rpcbind service, which returns a list of the current RPC     program-to-address mappings on <i>host</i>. It uses the transport specified through     <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL     if the remote rpcbind could not be contacted.      This interface returns all the mappings at the client's sensitivity label, and all     multilevel mappings.  rpcb_getallmaps()     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.</pre>

## rpcb\_getallmaps(3NSL)

### rpcb\_getaddr()

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3NSL)`.

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.

### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, `rpcb_gettime()` returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

### rpcb\_rmtcall()

An interface to the `rpcbind` service, which 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(3NSL)` 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.

Trusted Solaris 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 only available for connectionless transports.

### rpcb\_set()

An interface to the `rpcbind` service, which 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.

## rpcb\_getallmaps(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

### rpcb\_unset()

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

## 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`,  
`rpc_svc_calls(3NSL)`

`attributes(5)`

## rpcb\_getmaps(3NSL)

<b>NAME</b>	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
<b>SYNOPSIS</b>	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
<b>DESCRIPTION</b>	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
<b>Routines</b>	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()</pre> <p>An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL if the remote rpcbind could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>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.</p>

#### rpcb\_getaddr()

An interface to the rpcbind service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote rpcbind service. In the last case, the global variable *rpc\_createerr* contains the RPC status. See *rpc\_clnt\_create*(3NSL).

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.

#### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, *rpcb\_gettime()* returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

#### rpcb\_rmtcall()

An interface to the rpcbind service, which 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*(3NSL) 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.

Trusted Solaris 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 only available for connectionless transports.

#### rpcb\_set()

An interface to the rpcbind service, which 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.

## rpcb\_getmaps(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

### rpcb\_unset()

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`

`attributes(5)`



NAME	<p>rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service</p>
SYNOPSIS	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *rpcb_getmaps(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *rpcb_getallmaps(const struct netconfig     *netconf, const char *host);  bool_t rpcb_getaddr(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t rpcb_gettime(const char *host, time_t *timep);  enum clnt_stat rpcb_rmtcall(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t proct, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t rpcb_set(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t rpcb_unset(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
Routines	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()     An interface to the rpcbind service, which returns a list of the current RPC     program-to-address mappings on <i>host</i>. It uses the transport specified through     <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL     if the remote rpcbind could not be contacted.      This interface returns all the mappings at the client's sensitivity label, and all     multilevel mappings.  rpcb_getallmaps()     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.</pre>

## rpcb\_gettime(3NSL)

### rpcb\_getaddr()

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3NSL)`.

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.

### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, `rpcb_gettime()` returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

### rpcb\_rmtcall()

An interface to the `rpcbind` service, which 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(3NSL)` 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.

Trusted Solaris 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 only available for connectionless transports.

### rpcb\_set()

An interface to the `rpcbind` service, which 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.

rpcb\_gettime(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`,  
`rpc_svc_calls(3NSL)`

`attributes(5)`

## rpcbind(3NSL)

<b>NAME</b>	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
<b>SYNOPSIS</b>	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
<b>DESCRIPTION</b>	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
<b>Routines</b>	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()</pre> <p>An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL if the remote rpcbind could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>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.</p>

`rpcb_getaddr()`

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3NSL)`.

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.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `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.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which 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(3NSL)` 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.

Trusted Solaris 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 only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which 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.

## rpcbind(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

### `rpcb_unset()`

An interface to the `rpcbind` service, which 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-transport*] 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, 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(3NSL)`.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t proct, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
Routines	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()     An interface to the rpcbind service, which returns a list of the current RPC     program-to-address mappings on <i>host</i>. It uses the transport specified through     <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL     if the remote rpcbind could not be contacted.      This interface returns all the mappings at the client's sensitivity label, and all     multilevel mappings.  rpcb_getallmaps()     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.</pre>

## rpcb\_rmtcall(3NSL)

### rpcb\_getaddr()

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3NSL)`.

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.

### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, `rpcb_gettime()` returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

### rpcb\_rmtcall()

An interface to the `rpcbind` service, which 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(3NSL)` 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.

Trusted Solaris 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 only available for connectionless transports.

### rpcb\_set()

An interface to the `rpcbind` service, which 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.



rpcb\_rmtcall(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`,  
`rpc_svc_calls(3NSL)`

`attributes(5)`

## rpc\_broadcast(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

rpc\_broadcast(3NSL)

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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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*);

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const rpcproc\_t *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 *nettype* is NULL, it 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 it returns with appropriate status.

## rpc\_broadcast(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_broadcast_exp`(const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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*);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_call`(const char *host*, const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

	rpc_broadcast(3NSL)
	attribute-control structures and set the <code>t6attr_t</code> pointers in the <code>CLIENT</code> structure. When <code>clnt_destroy()</code> is used to destroy a client handle, the client should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that client handle.
Trusted Solaris 8 4/01 Reference Manual	rpc(3NSL), rpc_clnt_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL), t6free_blk(3NSL)
SunOS 5.8 Reference Manual	printf(3C), rpc_clnt_auth(3NSL), attributes(5)

## rpc\_broadcast\_exp(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

rpc\_broadcast\_exp(3NSL)

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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcrerror()` [see `rpc_clnt_create(3NSL)`], `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*);

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 rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const rpcproc\_t *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 *nettype* is NULL, it 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 it returns with appropriate status.

rpc\_broadcast\_exp(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_broadcast\_exp(const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat rpc\_call(const char \*host, const rpcprog\_t prognum, const rpcvers\_t versnum, const rpcproc\_t 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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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



rpc\_broadcast\_exp(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`,  
`t6free_blk(3NSL)`

`printf(3C)`, `rpc_clnt_auth(3NSL)`, `attributes(5)`

## rpcb\_set(3NSL)

<b>NAME</b>	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
<b>SYNOPSIS</b>	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
<b>DESCRIPTION</b>	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
<b>Routines</b>	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()</pre> <p>An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL if the remote rpcbind could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>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.</p>

## rpcb\_set(3NSL)

### rpcb\_getaddr()

An interface to the rpcbind service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote rpcbind service. In the last case, the global variable *rpc\_createerr* contains the RPC status. See *rpc\_clnt\_create(3NSL)*.

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.

### rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, *rpcb\_gettime()* returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

### rpcb\_rmtcall()

An interface to the rpcbind service, which 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(3NSL)* 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.

Trusted Solaris 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 only available for connectionless transports.

### rpcb\_set()

An interface to the rpcbind service, which 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.

rpcb\_set(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which 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-transport*] 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, 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(3NSL)`.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`,  
`rpc_svc_calls(3NSL)`

`attributes(5)`

rpcb\_unset(3NSL)

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include &lt;rpc/rpc.h&gt;  struct rpcblist *<b>rpcb_getmaps</b>(const struct netconfig *netconf,     const char *host);  struct tsol_rpcblist *<b>rpcb_getallmaps</b>(const struct netconfig     *netconf, const char *host);  bool_t <b>rpcb_getaddr</b>(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);  bool_t <b>rpcb_gettime</b>(const char *host, time_t *timep);  enum clnt_stat <b>rpcb_rmtcall</b>(const struct netconfig *netconf, const     char *host, const rpcprog_t prognum, const rpcvers_t versnum,     const rpcproc_t proct, const xdrproc_t inproc, const caddr_t     in, const xdrproc_t outproc, caddr_t out, const struct timeval     tout, struct netbuf *svcaddr);  bool_t <b>rpcb_set</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf, const struct netbuf *svcaddr);  bool_t <b>rpcb_unset</b>(const rpcprog_t prognum, const rpcvers_t versnum,     const struct netconfig *netconf);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M).</p>
Routines	<pre>#include &lt;rpc/rpc.h&gt;  rpcb_getmaps()     An interface to the rpcbind service, which returns a list of the current RPC     program-to-address mappings on <i>host</i>. It uses the transport specified through     <i>netconf</i> to contact the remote rpcbind service on <i>host</i>. This routine will return NULL     if the remote rpcbind could not be contacted.      This interface returns all the mappings at the client's sensitivity label, and all     multilevel mappings.  rpcb_getallmaps()     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.</pre>

rpcb\_unset(3NSL)

rpcb\_getaddr()

An interface to the rpcbind service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* 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 multilevel mapping exists, or that the RPC system failed to contact the remote rpcbind service. In the last case, the global variable *rpc\_createerr* contains the RPC status. See *rpc\_clnt\_create(3NSL)*.

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.

rpcb\_gettime()

This routine returns the time on *host* in *timep*. If *host* is NULL, *rpcb\_gettime()* returns the time on its own machine. This routine returns TRUE if it succeeds, 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.

rpcb\_rmtcall()

An interface to the rpcbind service, which 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(3NSL)* 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.

Trusted Solaris 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 only available for connectionless transports.

rpcb\_set()

An interface to the rpcbind service, which 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.

rpcb\_unset(3NSL)

This routine returns TRUE if it succeeds, FALSE otherwise. See also `svc_reg()` in `rpc_svc_calls(3NSL)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which 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-transport]` 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, 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(3NSL)`.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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 `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3NSL)`, `rpc_clnt_create(3NSL)`,  
`rpc_svc_calls(3NSL)`

`attributes(5)`

## rpc\_call(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>



```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

## rpc\_call(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_broadcast_exp`(const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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*);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_call`(const char *\*host*, const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

rpc\_call(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`,  
`t6free_blk(3NSL)`

**SunOS 5.8  
Reference Manual**

`printf(3C)`, `rpc_clnt_auth(3NSL)`, `attributes(5)`

## rpc\_clnt\_calls(3NSL)

<b>NAME</b>	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
<b>DESCRIPTION</b>	<p>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.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> 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).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const struct timeval tout);</p> <p>A function macro that 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 <code>clnt_create()</code> (see <code>rpc_clnt_create(3NSL)</code>). 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 of where 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 <code>clnt_control()</code>, see <code>rpc_clnt_create(3NSL)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t out);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</p> <p>A function macro that copies the error structure out of the client handle to the structure at address <i>errp</i>.</p>

```
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. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
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. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
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 which 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 if the programmer does not want the message to be output with `printf()` [see `printf(3C)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreaterror()` [see `rpc_clnt_create(3NSL)`], `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);
```

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 rpcprog_t prognum, const rpcvers_t versnum, const
rpcproc_t 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 *nettype* is NULL, it 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 it returns with appropriate status.

## rpc\_clnt\_calls(3NSL)

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(3NSL)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_broadcast_exp`(const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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*);

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 re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

enum clnt\_stat `rpc_call`(const char *\*host*, const `rpcprog_t` *prognum*, const `rpcvers_t` *versnum*, const `rpcproc_t` *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 of where to place the result(s). *nettype* can be any of the values listed on `rpc(3NSL)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. 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.

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

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

	rpc_clnt_calls(3NSL)
	attribute-control structures and set the <code>t6attr_t</code> pointers in the <code>CLIENT</code> structure. When <code>clnt_destroy()</code> is used to destroy a client handle, the client should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that client handle.
Trusted Solaris 8 4/01 Reference Manual	rpc(3NSL), rpc_clnt_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL), t6free_blk(3NSL)
SunOS 5.8 Reference Manual	printf(3C), rpc_clnt_auth(3NSL), attributes(5)

## rpc\_clnt\_create(3NSL)

<b>NAME</b>	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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
<b>DESCRIPTION</b>	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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</pre> <pre>CLGET_VERS    rpcvers_t    get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS    rpcvers_t    set the RPC program's version                                    number associated with the</pre>



client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID      uint32_t      get the XID of the previous
                    remote procedure call
CLSET_XID      uint32_t      set the XID of the next
                    remote procedure call
CLGET_PROG     rpcprog_t      get program number
CLSET_PROG     rpcprog_t      set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## rpc\_clnt\_create(3NSL)

routine is successful it 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(3NSL)*). 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.

```
CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t
*vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype const struct
timeval *timeout);
```

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

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

A function macro that 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 *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(3NSL)*).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf *svcaddr, const rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 in 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(3NSL)*). 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.

rpc\_clnt\_create(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## rpc\_clnt\_create(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

	rpc_clnt_create(3NSL)
	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.
Trusted Solaris 8 4/01 Reference Manual	rpcbind(1M), rpc(3NSL), rpc_clnt_calls(3NSL), rpc_svc_create(3NSL), libt6(3NSL), t6alloc_blk(3NSL), t6free_blk(3NSL)
SunOS 5.8 Reference Manual	rpc_clnt_auth(3NSL), svc_raw_create(3NSL), attributes(5)

## rpc\_createerr(3NSL)

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, 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
DESCRIPTION	<p>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.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>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.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the CLIENT data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info) ;</pre> <p>A function macro to change or retrieve 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, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval *      set total timeout CLGET_TIMEOUT  struct timeval *      get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code>, the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0, <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code>. Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf *    get server's address CLGET_SVC_ADDR    struct netbuf *    get server's address CLGET_FD          int *              get associated file descriptor 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        rpcvers_t          get the RPC program's version                                    number associated with the                                    client handle CLSET_VERS        rpcvers_t          set the RPC program's version                                    number associated with the</pre>

client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.

```
CLGET_XID    uint32_t    get the XID of the previous
                remote procedure call
CLSET_XID    uint32_t    set the XID of the next
                remote procedure call
CLGET_PROG   rpcprog_t   get program number
CLSET_PROG   rpcprog_t   set program number
```

The following 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 and FALSE on failure.

**CLIENT** `*clnt_create(const char *host, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where 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(3NSL)`).

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

Generic client creation routine which is similar to `clnt_create()` but which also 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 rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype);`

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the

## rpc\_createerr(3NSL)

routine is successful it 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(3NSL)*). 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.

CLIENT \*clnt\_create\_vers\_timed(const char \*host, const rpcprog\_t prognum, rpcvers\_t \*vers\_outp, const rpcvers\_t vers\_low, const rpcvers\_t vers\_high, char \*nettype const struct timeval \*timeout);

Generic client creation routine similar to *clnt\_create\_vers()* but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the *clnt\_create\_vers\_timed()* call behaves exactly like the *clnt\_create\_vers()* call.

void clnt\_destroy(CLIENT \*clnt) ;

A function macro that 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 *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(3NSL)*).

CLIENT \*clnt\_dg\_create(const int fildes, const struct netbuf \*svcaddr, const rpcprog\_t prognum, const rpcvers\_t versnum, const uint\_t sendsz, const uint\_t 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 in 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(3NSL)*). 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.



rpc\_createerr(3NSL)

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 rpcprog\_t *prognum*, const rpcvers\_t *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(3NSL)`). 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\_spcreateerror(const char \*s);

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 *fildev*, const struct netconfig \**netconf*, const struct netbuf \**svcaddr*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const uint\_t *sendsz*, const uint\_t *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 it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is NULL, a `RPC_UNKNOWNPROTO` error is set. If *fildev* 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(1M)`) is not consulted for the address of the remote service.

CLIENT \*clnt\_tp\_create(const char \**host*, const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const struct netconfig \**netconf*);

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

## rpc\_createerr(3NSL)

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 rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig *netconf, const struct timeval *timeout);
```

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 rpcprog_t
prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t 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 specify 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.

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;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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.

rpc\_createerr(3NSL)

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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_calls(3NSL)`, `rpc_svc_create(3NSL)`,  
`libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`

`rpc_clnt_auth(3NSL)`, `svc_raw_create(3NSL)`, `attributes(5)`

## rpc\_reg(3NSL)

<b>NAME</b>	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
<b>DESCRIPTION</b>	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-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 <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

rpc\_reg(3NSL)

If the server has the PRIV\_NET\_MAC\_READ privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the PRIV\_NET\_PRIVADDR privilege.

The PRIV\_NET\_SETID privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

int svc\_auth\_reg(const int cred\_flavor, const enum auth\_stat (\*handler)());

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

void xprt\_register(const SVCXPRT \*xprt);

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

void xprt\_unregister(const SVCXPRT \*xprt);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

## 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, `svc_unreg()` must be called with the privilege on.

rpc\_reg(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`

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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code>  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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## rpc\_svc\_calls(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.



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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval);
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

## rpc\_svc\_calls(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

### 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(3SOCKET) 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.

### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xprt\_register(3NSL), attributes(5)

### 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.

rpc\_svc\_calls(3NSL)

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## rpc\_svc\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

rpc\_svc\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const
rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);
    svc_create() creates server handles for all the transports belonging to the class
    nettype.
```

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

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

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

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

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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

## rpc\_svc\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

	rpc_svc_create(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

## rpc\_svc\_reg(3NSL)

<b>NAME</b>	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
<b>DESCRIPTION</b>	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-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 <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>



rpc\_svc\_reg(3NSL)

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

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

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

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

`void xprt_register(const SVCXPRT *xprt);`

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

`void xprt_unregister(const SVCXPRT *xprt);`

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### 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, `svc_unreg()` must be called with the privilege on.

rpc\_svc\_reg(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`

NAME	sbltos, sbcltos, sbsltos, sbclearartos – translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *sbcltos(const bclabel_t *label, const int len); char *sbsltos(const bslabel_t *label, const int len); char *sbclearartos(const bclear_t *clearance, const int len);</pre>
DESCRIPTION	<p>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's sensitivity label.</p> <p>These functions translate binary labels 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, "&lt;–", is appended to sensitivity labels and clearances. The character-coded label begins with a classification name separated with a single space character from the list of words making up the remainder of the label. The binary labels must be of the proper defined type and dominated by the process's sensitivity label. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbltos()</code> translates a binary CMW label into a clipped string. The function uses the long form of the words and the short form of the classification name, as in:</p> <pre>0xADMIN_LOW_hexadecimal_value [0xsensitivity_label_hexadecimal_value]</pre> <p>If <i>len</i> is less than the minimum number of characters (four), , the translation will fail.</p> <p><code>sbsltos()</code> translates a binary sensitivity label into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is less than the minimum number of characters (three), the translation fails.</p> <p><code>sbclearartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> 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 <code>label_encodings</code> file which may contain different words and constraints.</p>
RETURN VALUES	<p>These routines return a pointer to a statically allocated string that contains the result of the translation, or <code>(char *) 0</code> if the translation fails for any reason.</p>
<b>sbcltos</b>	<p>Assume that a CMW label is:</p> <pre>[UN TOP/MIDDLE/LOWER DRAWER]</pre>

sbclear(3TSOL)

when clipped to ten characters it is:

->[UN TOP/<-

**sbsl(3TSOL)**

Assume that a sensitivity label is:

UN TOP/MIDDLE/LOWER DRAWER

when clipped to ten characters it is:

UN TOP/M<-

**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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltcolor(3TSOL), blvalid(3TSOL), hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
WARNINGS**

attributes(5)

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	sbltos, sbcltos, sbsltos, sbcleartos – translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *sbcltos(const bclabel_t *label, const int len); char *sbsltos(const bslabel_t *label, const int len); char *sbcleartos(const bclear_t *clearance, const int len);</pre>
DESCRIPTION	<p>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's sensitivity label.</p> <p>These functions translate binary labels 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, "&lt;–", is appended to sensitivity labels and clearances. The character-coded label begins with a classification name separated with a single space character from the list of words making up the remainder of the label. The binary labels must be of the proper defined type and dominated by the process's sensitivity label. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbcltos()</code> translates a binary CMW label into a clipped string. The function uses the long form of the words and the short form of the classification name, as in:</p> <pre>0xADMIN_LOW_hexadecimal_value [0xsensitivity_label_hexadecimal_value]</pre> <p>If <i>len</i> is less than the minimum number of characters (four), , the translation will fail.</p> <p><code>sbsltos()</code> translates a binary sensitivity label into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is less than the minimum number of characters (three), the translation fails.</p> <p><code>sbcleartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> 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 <code>label_encodings</code> file which may contain different words and constraints.</p>
RETURN VALUES	<p>These routines return a pointer to a statically allocated string that contains the result of the translation, or <code>(char *) 0</code> if the translation fails for any reason.</p>
sbcltos	<p>Assume that a CMW label is:</p> <pre>[UN TOP/MIDDLE/LOWER DRAWER]</pre>

sbcltos(3TSOL)

when clipped to ten characters it is:

->[UN TOP/<-

**sbsltos**

Assume that a sensitivity label is:

UN TOP/MIDDLE/LOWER DRAWER

when clipped to ten characters it is:

UN TOP/M<-

**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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), blvalid(3TSOL), hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
WARNINGS**

attributes(5)

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	sbltos, sbcltos, sbsltos, sbcleartos – translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *sbcltos(const bclabel_t *label, const int len); char *sbsltos(const bslabel_t *label, const int len); char *sbcleartos(const bclear_t *clearance, const int len);</pre>
DESCRIPTION	<p>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's sensitivity label.</p> <p>These functions translate binary labels 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, "&lt;–", is appended to sensitivity labels and clearances. The character-coded label begins with a classification name separated with a single space character from the list of words making up the remainder of the label. The binary labels must be of the proper defined type and dominated by the process's sensitivity label. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p>sbccltos() translates a binary CMW label into a clipped string. The function uses the long form of the words and the short form of the classification name, as in:</p> <pre>0xADMIN_LOW_hexadecimal_value [0xsensitivity_label_hexadecimal_value]</pre> <p>If <i>len</i> is less than the minimum number of characters (four), , the translation will fail.</p> <p>sbsltos() translates a binary sensitivity label into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is less than the minimum number of characters (three), the translation fails.</p> <p>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 <i>len</i> 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.</p>
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.
sbcltos	<p>Assume that a CMW label is:</p> <pre>[UN TOP/MIDDLE/LOWER DRAWER]</pre>

sbltos(3TSOL)

when clipped to ten characters it is:

->[UN TOP/<-

**sbsltos**

Assume that a sensitivity label is:

UN TOP/MIDDLE/LOWER DRAWER

when clipped to ten characters it is:

UN TOP/M<-

**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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolo(3TSOL), blvalid(3TSOL), hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
WARNINGS**

attributes(5)

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	sbltos, sbcltos, sbsltos, sbcleartos – translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  char *sbcltos(const bclabel_t *label, const int len); char *sbsltos(const bslabel_t *label, const int len); char *sbcleartos(const bclear_t *clearance, const int len);</pre>
DESCRIPTION	<p>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's sensitivity label.</p> <p>These functions translate binary labels 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, "&lt;–", is appended to sensitivity labels and clearances. The character-coded label begins with a classification name separated with a single space character from the list of words making up the remainder of the label. The binary labels must be of the proper defined type and dominated by the process's sensitivity label. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p>sbltos() translates a binary CMW label into a clipped string. The function uses the long form of the words and the short form of the classification name, as in:</p> <pre>0xADMIN_LOW_hexadecimal_value [0xsensitivity_label_hexadecimal_value]</pre> <p>If <i>len</i> is less than the minimum number of characters (four), , the translation will fail.</p> <p>sbsltos() translates a binary sensitivity label into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is less than the minimum number of characters (three), the translation fails.</p> <p>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 <i>len</i> 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.</p>
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.
sbcltos	<p>Assume that a CMW label is:</p> <pre>[UN TOP/MIDDLE/LOWER DRAWER]</pre>

sbsltos(3TSOL)

when clipped to ten characters it is:

```
->[UN TOP/<-
```

**sbsltos**

Assume that a sensitivity label is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<-
```

**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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), blvalid(3TSOL), hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
WARNINGS**

attributes(5)

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 [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  ssize_t send(int s, const void *msg, size_t len, int flags); ssize_t sendto(int s, const void *msg, size_t len, int flags, const                struct sockaddr *to, int tolen); ssize_t sendmsg(int s, const struct msghdr *msg, int flags);</pre>				
DESCRIPTION	<p>send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3SOCKET).</p> <p>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>. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (seefcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families 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. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See recv(3SOCKET), for a description of the msghdr structure.</p>	MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td>EBADF</td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	EBADF	<i>s</i> is an invalid file descriptor.		
EBADF	<i>s</i> is an invalid file descriptor.				

send(3SOCKET)

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to 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 non-blocking and the requested operation would block.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

fcntl(2), write(2), getsockopt(3SOCKET), socket(3SOCKET)  
poll(2), select(3C), socket(3HEAD), recv(3SOCKET), attributes(5)

NAME	send, sendto, sendmsg – send a message from a socket				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  ssize_t send(int s, const void *msg, size_t len, int flags); ssize_t sendto(int s, const void *msg, size_t len, int flags, const                struct sockaddr *to, int tolen); ssize_t sendmsg(int s, const struct msghdr *msg, int flags);</pre>				
DESCRIPTION	<p>send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3SOCKET).</p> <p>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>. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (seefcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families 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. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See recv(3SOCKET), for a description of the msghdr structure.</p>	MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td>EBADF</td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	EBADF	<i>s</i> is an invalid file descriptor.		
EBADF	<i>s</i> is an invalid file descriptor.				

sendmsg(3SOCKET)

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len 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 non-blocking and the requested operation would block.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

fcntl(2), write(2), getsockopt(3SOCKET), socket(3SOCKET)  
poll(2), select(3C), socket(3HEAD), recv(3SOCKET), attributes(5)

NAME	send, sendto, sendmsg – send a message from a socket				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  ssize_t send(int s, const void *msg, size_t len, int flags); ssize_t sendto(int s, const void *msg, size_t len, int flags, const                struct sockaddr *to, int tolen); ssize_t sendmsg(int s, const struct msghdr *msg, int flags);</pre>				
DESCRIPTION	<p>send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3SOCKET).</p> <p>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>. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (seefcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families 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. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See recv(3SOCKET), for a description of the msghdr structure.</p>	MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address families support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td>EBADF</td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	EBADF	<i>s</i> is an invalid file descriptor.		
EBADF	<i>s</i> is an invalid file descriptor.				

sendto(3SOCKET)

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len 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 non-blocking and the requested operation would block.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

fcntl(2), write(2), getsockopt(3SOCKET), socket(3SOCKET)  
poll(2), select(3C), socket(3HEAD), recv(3SOCKET), attributes(5)



<b>NAME</b>	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;bsm/libbsm.h&gt;  int getacdir( char *dir, int len);  int getacmin( int *min_val);  int getacflg( char *auditstring, int len);  int getacna( char *auditstring, int len);  void setac( void);  void endac( void);</pre>		
<b>DESCRIPTION</b>	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> 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 <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
<b>FILES</b>	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
<b>RETURN VALUES</b>	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <ul style="list-style-type: none"> <li>0           on success.</li> <li>-2           On failure and set <code>errno</code> to indicate the error.</li> </ul> <p><code>getacmin()</code> and <code>getacflg()</code> return:</p> <ul style="list-style-type: none"> <li>1           On EOF.</li> </ul> <p><code>getacdir()</code> returns:</p>		

setac(3BSM)

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 If the directory entry format in the `audit_control` file is incorrect.

`getacdir()`, `getacflg()` and `getacna()` return:

- 3 If the input buffer is too short to accommodate the record.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_warn(1M)`, `inetd(1M)`, `audit_control(4)`

`attributes(5)`

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get_audit_class entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_class_ent *<b>getauclassnam</b>( const char *<i>name</i> ); struct au_class_ent *<b>getauclassnam_r</b>( au_class_ent_t *<i>class_int</i>,     const char *<i>name</i> ); struct au_class_ent *<b>getauclassent</b>( void ); struct au_class_ent *<b>getauclassent_r</b>( au_class_ent_t *<i>class_int</i> ); void <b>setauclass</b>( void ); void <b>endauclass</b>( void ); </pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent().</p> <p>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.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure 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 ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in &lt;bsm/libbsm.h&gt; with the following members:</p> <pre> char          *ac_name; au_class_t    ac_class; char          *ac_desc; </pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

setauclass(3BSM)

getauclassent() and getauclassent\_r() return a pointer to a struct au\_class\_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

**FILES**

/etc/security/audit\_class      Maps audit class numbers to audit class names.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
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.

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

audit\_class(4), audit\_event(4)

attributes(5)

**NOTES**

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	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre> <b>cc</b> [<i>flags...</i>] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library...</i>]  #include &lt;sys/param.h&gt; #include &lt;bsm/libbsm.h&gt;  struct au_event_ent *getauevent(void);  struct au_event_ent *getauevnam(char *name);  struct au_event_ent *getauevnum(au_event_t event_number);  au_event_t *getauevnonam(char *event_name);  void setauevent(void);  void endauevent(void);  struct au_event_ent *getauevent_r(au_event_ent_t *e);  struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name);  struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t     event_number); </pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() 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.</p>

setauevent(3BSM)

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 assure there is enough space for the information returned, the applications programmer should be sure to 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 an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

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

## RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

## FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

## SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`getauclassent(3BSM)`, `audit_class(4)`, `audit_event(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

setauevent(3BSM)

**NOTES** All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

## setauthattr(3SECDB)

NAME	getauthattr, getauthnam, free_authattr, setauthattr, endauthattr, chkauthattr – get authorization entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;auth_attr.h&gt; #include &lt;secdb.h&gt;  authattr_t *getauthattr(void);  authattr_t *getauthnam(const char *name);  void free_authattr(authattr_t *auth);  void setauthattr(void);  void endauthattr(void);  int chkauthattr(const char *authname, const char *username);</pre>
DESCRIPTION	<p>The <code>getauthattr()</code> and <code>getauthnam()</code> functions each return an <code>auth_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getauthattr()</code> function enumerates <code>auth_attr</code> entries. The <code>getauthnam()</code> function searches for an <code>auth_attr</code> entry with a given authorization name <i>name</i>. Successive calls to these functions return either successive <code>auth_attr</code> entries or NULL.</p> <p>The internal representation of an <code>auth_attr</code> entry is an <code>authattr_t</code> structure defined in <code>&lt;auth_attr.h&gt;</code> with the following members:</p> <pre>char  *name;           /* name of the authorization */ char  *res1;           /* reserved for future use */ char  *res2;           /* reserved for future use */ char  *short_desc;     /* short description */ char  *long_desc;      /* long description */ kva_t *attr;           /* array of key-value pair attributes */</pre> <p>The <code>setauthattr()</code> function “rewinds” to the beginning of the enumeration of <code>auth_attr</code> entries. Calls to <code>getauthnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setauthattr()</code> should be called before the first call to <code>getauthattr()</code>.</p> <p>The <code>endauthattr()</code> function may be called to indicate that <code>auth_attr</code> processing is complete; the system may then close any open <code>auth_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>chkauthattr()</code> function verifies whether or not a user has a given authorization. It first reads the <code>AUTHS_GRANTED</code> key in the <code>/etc/security/policy.conf</code> file and returns 1 if it finds a match for the given authorization. If <code>chkauthattr()</code> does not find a match, it reads the <code>PROFS_GRANTED</code> key in <code>/etc/security/policy.conf</code> and returns 1 if the given authorization is in any profiles specified with the <code>PROFS_GRANTED</code> keyword. If a</p>



match is not found from the default authorizations and default profiles, `chkauthattr()` reads the `user_attr(4)` database. If it does not find a match in `user_attr`, it reads the `prof_attr(4)` database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The `chkauthattr()` function returns 0 if it does not find a match in any of the three sources.

A user is considered to have been assigned an authorization if either of the following are true:

- The authorization name matches exactly any authorization assigned in the `user_attr` or `prof_attr` databases (authorization names are case-sensitive).
- The authorization name suffix is not the keyword `grant` and the authorization name matches any authorization up to the asterisk (\*) character assigned in the `user_attr` or `prof_attr` databases.

The examples in the following table illustrate the conditions under which a user is assigned an authorization.

	<code>/etc/security/policy.conf</code> or	Is user
Authorization name	<code>user_attr</code> or <code>prof_attr</code> entry	authorized?
<code>solaris.printer.postscript</code>	<code>solaris.printer.postscript</code>	Yes
<code>solaris.printer.postscript</code>	<code>solaris.printer.*</code>	Yes
<code>solaris.printer.grant</code>	<code>solaris.printer.*</code>	No

The `free_authattr()` function releases memory allocated by the `getauthnam()` and `getauthattr()` functions.

## RETURN VALUES

The `getauthattr()` function returns a pointer to an `authattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The `getauthnam()` function returns a pointer to an `authattr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `chkauthattr()` function returns 1 if the user is authorized and 0 otherwise.

## USAGE

The `getauthattr()` and `getauthnam()` functions both allocate memory for the pointers they return. This memory should be de-allocated with the `free_authattr()` call.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

setauthattr(3SECDB)

Individual attributes in the `attr` structure can be referred to by calling the `kva_match(3SECDB)` function.

#### WARNINGS

Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

#### FILES

<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch
<code>/etc/user_attr</code>	extended user attributes
<code>/etc/security/auth_attr</code>	authorization attributes
<code>/etc/security/policy.conf</code>	policy definitions
<code>/etc/security/prof_attr</code>	profile information

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The Trusted Solaris environment adds authorizations. The `chkauthattr()` function replaces the Trusted Solaris 7 `chkauth()` function.

`nsswitch.conf(4)`, `prof_attr(4)`, `user_attr(4)`

`getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `getuserattr(3SECDB)`,  
`kva_match(3SECDB)`, `auth_attr(4)`, `attributes(5)`, `rbac(5)`

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flag...] file ... -lbsm -lsocket -lnsl -lintl [library...]  #include &lt;sys/param.h&gt;  #include &lt;bsm/libbsm.h&gt;  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_t *u, const char     *name); struct au_user_ent *getauuserent_r(au_user_ent_t *u);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry. Entries can come from any of the sources specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code>&lt;bsm/libbsm.h&gt;</code> with the following members:</p> <pre>char      *au_name; au_mask_t au_always; au_mask_t au_never;</pre>

setauuser(3BSM)

## RETURN VALUES

The `getauusernam()` function returns a pointer to a `struct_au_user_ent` if it successfully locates the requested entry; otherwise it returns `NULL`.

The `getauuserent()` function returns a pointer to a `struct_au_user_ent` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

## FILES

`/etc/security/audit_user` Stores per-user audit event mask.  
`/etc/passwd` Stores user-id to username mappings.

## SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8  
4/01 Reference  
Manual  
Solaris 9  
Reference Manual

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

`audit_user(4)`, `nsswitch.conf(4)`

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

## NOTES

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

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	bltype, setbltype – compare and set the type of binary label										
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  int <b>bltype</b>(const void *label, const unsigned char type); void <b>setbltype</b>(void *label, const unsigned char type);</pre>										
DESCRIPTION	<p>These functions compare and set the type of binary labels.</p> <p><code>bltype()</code> examines <i>label</i> to determine if it is of the specified type <i>type</i>.</p> <p><code>setbltype()</code> sets the type of <i>label</i> to the specified type <i>type</i>.</p> <p><i>type</i> may be one of:</p> <table><tr><td>SUN_SL_ID</td><td><i>label</i> is a defined binary sensitivity label.</td></tr><tr><td>SUN_SL_UN</td><td><i>label</i> is an undefined binary sensitivity label.</td></tr><tr><td>SUN_CLR_ID</td><td><i>label</i> is a defined binary clearance.</td></tr><tr><td>SUN_CLR_UN</td><td><i>label</i> is an undefined binary clearance.</td></tr><tr><td>SUN_CMW_ID</td><td><i>label</i> is a binary CMW label whose label portions may or may not be defined. (<code>bltype()</code> only.)</td></tr></table>	SUN_SL_ID	<i>label</i> is a defined binary sensitivity label.	SUN_SL_UN	<i>label</i> is an undefined binary sensitivity label.	SUN_CLR_ID	<i>label</i> is a defined binary clearance.	SUN_CLR_UN	<i>label</i> is an undefined binary clearance.	SUN_CMW_ID	<i>label</i> is a binary CMW label whose label portions may or may not be defined. ( <code>bltype()</code> only.)
SUN_SL_ID	<i>label</i> is a defined binary sensitivity label.										
SUN_SL_UN	<i>label</i> is an undefined binary sensitivity label.										
SUN_CLR_ID	<i>label</i> is a defined binary clearance.										
SUN_CLR_UN	<i>label</i> is an undefined binary clearance.										
SUN_CMW_ID	<i>label</i> is a binary CMW label whose label portions may or may not be defined. ( <code>bltype()</code> only.)										
RETURN VALUES	<code>bltype()</code> returns non-zero if <i>label</i> is of type <i>type</i> , otherwise zero is returned.										
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Availability	SUNWtsu										
MT-Level	MT-Safe										
Trusted Solaris 8 4/01 Reference Manual	<p><code>bcltobanner(3TSOL)</code>, <code>blcompare(3TSOL)</code>, <code>bltocolor(3TSOL)</code>, <code>btohex(3TSOL)</code>, <code>labelinfo(3TSOL)</code></p> <p><i>Trusted Solaris Developer's Guide</i></p>										
SunOS 5.8 Reference Manual WARNINGS	<p><code>attributes(5)</code></p> <ol style="list-style-type: none"><li><code>bltype(&amp;cmw_label, SUN_CMW_ID)</code> checks the existence of a binary CMW label structure and not the portions of the structure that contain defined labels.</li><li>When attempting to determine the type of a label, rather than to verify that a specific label type is present, check <code>SUN_CMW_ID</code> first.</li><li><code>setbltype()</code> makes no checks on the structure it is setting or the type value.</li></ol>										

setcs1(3TSOL)

NAME	blportion, bcltos1, getcs1, setcs1 – access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]  #include &lt;tsol/label.h&gt;  bslabel_t *bcltos1(bslabel_t *label) ;  void getcs1(bslabel_t *destination_label, const bslabel_t *source_label) ; void setcs1(bslabel_t *destination_label, const bslabel_t *source_label) ;</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltos1() provides a pointer to the sensitivity label of the binary CMW label <i>label</i>.</p> <p>getcs1() copies the sensitivity label of the binary CMW label <i>source_label</i> to the binary sensitivity label <i>destination_label</i>.</p> <p>setcs1() replaces the value of the sensitivity label of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label <i>source_label</i>.</p>						
RETURN VALUES	bcltos1() returns a pointer to its label type.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p><b>EXAMPLE 1</b> Comparing Sensitivity Labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blequal(bcltos1(&amp;cmw_label), &amp;file_sensitivity_label)</pre>						
Trusted Solaris 8 4/01 Reference Manual	<pre>bcltobanner(3TSOL), blcompare(3TSOL), bltos(3TSOL), btohex(3TSOL), labelinfo(3TSOL)</pre> <p><i>Trusted Solaris Developer's Guide</i></p>						
SunOS 5.8 Reference Manual	<pre>attributes(5)</pre>						

set\_effective\_priv(3TSOL)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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 <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <table> <tr> <td>PRIV_ON</td><td>Add the specified privilege IDs to the privilege set of the target process.</td></tr> <tr> <td>PRIV_OFF</td><td>Clear the specified privileges from the privilege set of the target process.</td></tr> <tr> <td>PRIV_SET</td><td>Add the specified privilege IDs to the privilege set of the target process and clear all other privileges.</td></tr> </table> <p><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 <code>&lt;priv_names.h&gt;</code>.</p> <p>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.</p>	PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.	PRIV_OFF	Clear the specified privileges 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.
PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.						
PRIV_OFF	Clear the specified privileges 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.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set <code>errno</code> to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set <code>errno</code> to indicate the error.		
0	On success.						
-1	On failure, and set <code>errno</code> to indicate the error.						
ERRORS	<table> <tr> <td>EINVAL</td><td>The specified privilege is invalid.</td></tr> </table>	EINVAL	The specified privilege is invalid.				
EINVAL	The specified privilege is invalid.						

set\_effective\_priv(3TSOL)

EPERM

A specified privilege is not permitted in the asserted set of privileges.

**Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual**

setppriv(2)

attributes(5)



NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [ <i>flag...</i> ] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [ <i>library...</i> ] #include &lt;exec_attr.h&gt; #include &lt;secdb.h&gt;  execattr_t *getexecattr(void);  void free_execattr(execattr_t *ep);  void setexecattr(void);  void endexecattr(void);  execattr_t *getexecuser(const char *username, const char *type,                         const char *id, int search_flag);  execattr_t *getexecprof(const char *profname, const char *type,                         const char *id, int search_flag);  execattr_t *match_execattr(execattr_t *ep, char *profname, char                            *type, char *id);</pre>
DESCRIPTION	<p>The <code>getexecattr()</code> function returns a single <code>exec_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or NULL. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to NULL.</p> <p>The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code>&lt;exec_attr.h&gt;</code> with the following members:</p> <pre>char          name;    /* name of the profile */ char          type;    /* type of profile */ char          policy;  /* policy under which the attributes are */                 /* relevant*/ char          res1;    /* reserved for future use */ char          res2;    /* reserved for future use */ char          id;      /* unique identifier */ kva_t         attr;    /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> <p>The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released.</p> <p>The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>.</p>

## setexecattr(3SECDB)

The `endexecattr()` function can be called to indicate that `exec_attr` processing is complete; the library can then close any open `exec_attr` file, deallocate any internal storage, and so forth.

The `getexecuser()` function returns a linked list of entries filtered by the function's arguments. Only entries assigned to the specified *username*, as described in the `passwd(4)` database, and containing the specified *type* and *id*, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function's arguments. Only entries in the database matching the argument *profname*, as described in `exec_attr`, and containing the *type* and *id*, also described in `exec_attr`, are placed in the list.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See `EXAMPLES`.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

### RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

### USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

EXAMPLES

**EXAMPLE 1** The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
    GET_ONE)) == NULL) {
    /* do error */
}
```

**EXAMPLE 2** The following finds the entry for the `ping` command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 3** The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

**EXAMPLE 4** The following tells if the `tar` command is in a profile assigned to user `wetmore`. If there is no exact profile entry, the wildcard (`*`), if defined, is returned.

```
if ((execprof=getexecprof("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

FILES

- `/etc/nsswitch.conf` configuration file lookup information for the name server switch
- `/etc/user_attr` extended user attributes
- `/etc/security/exec_attr` execution profiles
- `/etc/security/policy.conf` policy definitions

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

setexecattr(3SECDB)

**SEE ALSO** getauthattr(3SECDB), getuserattr(3SECDB), kva\_match(3SECDB),  
exec\_attr(4), policy.conf(4), user\_attr(4), attributes(5)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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 <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <table> <tr> <td>PRIV_ON</td><td>Add the specified privilege IDs to the privilege set of the target process.</td></tr> <tr> <td>PRIV_OFF</td><td>Clear the specified privileges from the privilege set of the target process.</td></tr> <tr> <td>PRIV_SET</td><td>Add the specified privilege IDs to the privilege set of the target process and clear all other privileges.</td></tr> </table> <p><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 <code>&lt;priv_names.h&gt;</code>.</p> <p>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.</p>	PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.	PRIV_OFF	Clear the specified privileges 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.
PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.						
PRIV_OFF	Clear the specified privileges 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.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set <code>errno</code> to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set <code>errno</code> to indicate the error.		
0	On success.						
-1	On failure, and set <code>errno</code> to indicate the error.						
ERRORS	<table> <tr> <td>EINVAL</td><td>The specified privilege is invalid.</td></tr> </table>	EINVAL	The specified privilege is invalid.				
EINVAL	The specified privilege is invalid.						

set\_inheritable\_priv(3TSOL)

EPERM

A specified privilege is not permitted in the asserted set of privileges.

**Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 4  
Reference Manual**

setppriv(2)

attributes(5)

set\_permitted\_priv(3TSOL)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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 <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <table><tr><td>PRIV_ON</td><td>Add the specified privilege IDs to the privilege set of the target process.</td></tr><tr><td>PRIV_OFF</td><td>Clear the specified privileges from the privilege set of the target process.</td></tr><tr><td>PRIV_SET</td><td>Add the specified privilege IDs to the privilege set of the target process and clear all other privileges.</td></tr></table> <p><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 <code>&lt;priv_names.h&gt;</code>.</p> <p>Note that if <i>op</i> is <code>PRIV_SET</code> and <i>privno</i> is 0, the target privilege set is initialized to the empty set.</p>	PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.	PRIV_OFF	Clear the specified privileges 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.
PRIV_ON	Add the specified privilege IDs to the privilege set of the target process.						
PRIV_OFF	Clear the specified privileges 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.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and set <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and set <code>errno</code> to indicate the error.		
0	On success.						
-1	On failure, and set <code>errno</code> to indicate the error.						
ERRORS	<table><tr><td>EINVAL</td><td>The specified privilege is invalid.</td></tr></table>	EINVAL	The specified privilege is invalid.				
EINVAL	The specified privilege is invalid.						

set\_permitted\_priv(3TSOL)

EPERM

A specified privilege is not permitted in the asserted set of privileges.

**Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual**

setppriv(2)

attributes(5)



NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [ flag... ] file... -lsecdB -lsocket -lnsl -lintl [ library... ] #include &lt;prof.h&gt;  profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	<p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions each return a <code>prof_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file.</p> <p>The <code>getprofattr()</code> function enumerates <code>prof_attr</code> entries. The <code>getprofnam()</code> function searches for a <code>prof_attr</code> entry with a given <i>name</i>. Successive calls to these functions return either successive <code>prof_attr</code> entries or NULL.</p> <p>The internal representation of a <code>prof_attr</code> entry is a <code>profattr_t</code> structure defined in <code>&lt;prof_attr.h&gt;</code> with the following members:</p> <pre>char    name;    /* Name of the profile */ char    res1;    /* Reserved for future use */ char    res2;    /* Reserved for future use */ char    desc;    /* Description/Purpose of the profile */ kva_t   attr;    /* Profile attributes */</pre> <p>The <code>free_profattr()</code> function releases memory allocated by the <code>getprofattr()</code> and <code>getprofnam()</code> functions.</p> <p>The <code>setprofattr()</code> function “rewinds” to the beginning of the enumeration of <code>prof_attr</code> entries. Calls to <code>getprofnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setprofattr()</code> should be called before the first call to <code>getprofattr()</code>.</p> <p>The <code>endprofattr()</code> function may be called to indicate that <code>prof_attr</code> processing is complete; the system may then close any open <code>prof_attr</code> file, deallocate storage, and so forth.</p> <p>The <code>getproflist()</code> function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by <code>getproflist()</code> should be freed using the <code>free_proflist()</code> function.</p>

setprofattr(3SECDB)

	<p>The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.</p>				
RETURN VALUES	<p>The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>				
USAGE	<p>Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function.</p> <p>Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error.</p> <p>The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function.</p> <p>Applications 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. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.</p>				
FILES	<code>/etc/security/prof_attr</code> profiles and their descriptions				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>auths(1)</code> , <code>profiles(1)</code> , <code>getexecattr(3SECDB)</code> , <code>getauthattr(3SECDB)</code> , <code>prof_attr(4)</code>				

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get user profile description
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsoldb -lcmd -lnsl [ <i>library...</i> ] ( <b>obsolete</b> )
DESCRIPTION	The getprofent, setprofent, endprofent, getprofentbyname, and free_profent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

setprofstr(3TSOL)

<b>NAME</b>	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get user profile description
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, and free_profstr functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getprofattr(3SECDB) and getexecattr(3SECDB) man pages. These functions find rights profiles information in prof_attr(4) and exec_attr(4).

NAME	getsockopt, setsockopt – get and set options on sockets
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int <b>getsockopt</b>(int s, int level, int optname, void *optval, int *optlen); int <b>setsockopt</b>(int s, int level, int optname, const void *optval, int                optlen);</pre>
DESCRIPTION	<p>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.</p> <p>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(3SOCKET)).</p> <p>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.</p> <p><i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file &lt;sys/socket.h&gt; contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</p> <p>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 &lt;sys/socket.h&gt;. struct linger contains the following members:</p> <pre>l_onoff          on = 1/off = 0 l_linger         linger time, in seconds</pre> <p>The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt().</p> <pre>SO_DEBUG         enable/disable recording of debugging information SO_REUSEADDR     enable/disable local address reuse SO_KEEPALIVE     enable/disable keep connections alive</pre>

## setsockopt(3SOCKET)

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_DGRAM_ERRIND	application wants delayed error
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(3SOCKET)` 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. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

setsockopt(3SOCKET)

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. `SO_DGRAM_ERRIND` makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (*to* parameter) to fail with the appropriate error. See `send(3SOCKET)`.

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.

## RETURN VALUES

`getsockopt()` returns:

0            On success.

-1           On failure, and sets `errno` to indicate the error.

## ERRORS

The call succeeds unless:

`EBADF`                            The argument *s* 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.

`ENOSR`                            There were insufficient STREAMS resources available for the operation to complete.

`ENOTSOCK`                        The argument *s* is not a socket.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

## 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
Sum 39-53  
Reference Manual

`read(2)`, `bind(3SOCKET)`, `send(3SOCKET)`, `socket(3SOCKET)`

`close(2)`, `ioctl(2)`, `getprotobyname(3SOCKET)`, `recv(3SOCKET)`, `netconfig(4)`, `attributes(5)`

## setuserattr(3SECDB)

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [ flag... ] file... - lsecdb - lsocket - lns1 - lint1 [ library... ] #include &lt;user_attr.h&gt;  userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	<p>The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL.</p> <p>The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions.</p> <p>The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code>&lt;user_attr.h&gt;</code> with the following members:</p> <pre>char    name;      /* name of the user */ char    qualifier; /* reserved for future use */ char    res1;      /* reserved for future use */ char    res2;      /* reserved for future use */ kva_t    attr;     /* list of attributes */</pre> <p>The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>.</p> <p>The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.</p>
RETURN VALUES	<p>The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration.</p> <p>The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.</p>



**USAGE** The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications 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. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

**WARNINGS** Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

**FILES**

<code>/etc/user_attr</code>	extended user attributes
<code>/etc/nsswitch.conf</code>	configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

setuserent(3TSOL)

<b>NAME</b>	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get user security attributes
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsolddb -ltsol -lnsl -lcmd [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	The getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, and free_userent functions are replaced in Trusted Solaris 8 and later releases with the functions described in the getuserattr(3SECDB) man page. These functions find user security attributes in user_attr(4).

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];    /* user login name */ char          ut_id[4];     /* /sbin/inittab id (usually line #) */ char          ut_line[12];  /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;            /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

## setutent(3C)

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

	setutent(3C)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	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.
<b>SunOS 5.8 Reference Manual</b>	ttyslot(3C), utmp(4), utmpx(4), attributes(5)
<b>NOTES</b>	<p>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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().</p>

## setutxent(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## setutxent(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>



The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference**

**Manual**

**Reference Manual**

**NOTES**

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## socket(3SOCKET)

NAME	socket – create an endpoint for communication						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt;  int socket(int domain, int type, int protocol);</pre>						
DESCRIPTION	<p>socket () creates an endpoint for communication and returns a descriptor.</p> <p>The <i>domain</i> parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file &lt;sys/socket.h&gt;. There must be an entry in the netconfig(4) file for at least each protocol family and type required. If <i>protocol</i> has been specified, but no exact match for the tuple family, type, protocol is found, then the first entry containing the specified family and type with zero for protocol will be used. The currently understood formats are:</p> <table> <tr> <td>PF_UNIX</td><td>UNIX system internal protocols</td></tr> <tr> <td>PF_INET</td><td>Internet Protocol Version 4 (IPv4)</td></tr> <tr> <td>PF_INET6</td><td>Internet Protocol Version 6 (IPv6)</td></tr> </table> <p>The socket has the indicated <i>type</i>, which specifies the communication semantics. Currently defined types are:</p> <pre>SOCK_STREAM SOCK_DGRAM SOCK_RAW SOCK_SEQPACKET SOCK_RDM</pre> <p>A SOCK_STREAM type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A SOCK_SEQPACKET socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently not implemented for any protocol family. SOCK_RAW sockets provide access to internal network interfaces. The types SOCK_RAW, which is available only to a process with the net_rawaccess privilege, and SOCK_RDM, for which no implementation currently exists, are not described here.</p> <p><i>protocol</i> specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, multiple protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the</p>	PF_UNIX	UNIX system internal protocols	PF_INET	Internet Protocol Version 4 (IPv4)	PF_INET6	Internet Protocol Version 6 (IPv6)
PF_UNIX	UNIX system internal protocols						
PF_INET	Internet Protocol Version 4 (IPv4)						
PF_INET6	Internet Protocol Version 6 (IPv6)						

“communication domain” in which communication is to take place. If a protocol is specified by the caller, then it will be packaged into a socket level option request and sent to the underlying protocol layers.

Sockets of type `SOCK_STREAM` are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a `connect(3SOCKET)` call. Once connected, data may be transferred using `read(2)` and `write(2)` calls or some variant of the `send(3SOCKET)` and `recv(3SOCKET)` calls. When a session has been completed, a `close(2)` may be performed. Out-of-band data may also be transmitted as described on the `send(3SOCKET)` manual page and received as described on the `recv(3SOCKET)` manual page.

The communications protocols used to implement a `SOCK_STREAM` insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with `-1` returns and with `ETIMEDOUT` as the specific code in the global variable `errno`. The protocols optionally keep sockets “warm” by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (for instance 5 minutes). A `SIGPIPE` signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

`SOCK_SEQPACKET` sockets employ the same system calls as `SOCK_STREAM` sockets. The only difference is that `read(2)` calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

`SOCK_DGRAM` and `SOCK_RAW` sockets allow datagrams to be sent to correspondents named in `sendto(3SOCKET)` calls. Datagrams are generally received with `recvfrom(3SOCKET)`, which returns the next datagram with its return address.

An `fcntl(2)` call can be used to specify a process group to receive a `SIGURG` signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events with `SIGIO` signals.

The operation of sockets is controlled by socket level *options*. These options are defined in the file `<sys/socket.h>`. `setsockopt(3SOCKET)` and `getsockopt(3SOCKET)` are used to set and get options, respectively.

## RETURN VALUES

A `-1` is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

## ERRORS

The `socket()` call fails if:

`EACCES`

Permission to create a socket of the specified type and/or protocol is denied.

`EMFILE`

The per-process descriptor table is full.

socket(3SOCKET)

ENOMEM	Insufficient user memory is available.
ENOSR	There were insufficient STREAMS resources available to complete the operation.
EPROTONOSUPPORT	The protocol type or the specified protocol is not supported within this domain.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY OF TRUSTED SOLARIS CHANGES**  
Trusted Solaris 8 4/01 Reference Manual

The SOCK\_RAW socket is available only to a process with the net\_rawaccess privilege.

fcntl(2), read(2), write(2), accept(3SOCKET), bind(3SOCKET), getsockopt(3SOCKET), listen(3SOCKET), setsockopt(3SOCKET), send(3SOCKET), attributes(5)

**SunOS 5.8 Reference Manual**

close(2), ioctl(2), in(3HEAD), socket(3HEAD), connect(3SOCKET), getsockname(3SOCKET), recv(3SOCKET), shutdown(3SOCKET), socketpair(3SOCKET), attributes(5)

NAME	stobl, stobcl, stobsl, stobclear – translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  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 stobclear(const char *string, bclear_t *clearance, const int flags,               int *error) ;</pre>				
DESCRIPTION	<p>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's sensitivity label.</p> <p>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.</p> <p>The generic form of an input character-coded label string is:</p> <pre>[ [ + ] classification name ] [ [ +   - ] word ... ]</pre> <p>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 classification name followed by a + or –, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels (see <code>hextob()</code>) when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.</td></tr> </table>	NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.	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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.
NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.				
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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.				

stobcl(3TSOL)

0 (zero)

The default action is taken.

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

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

[*sensitivity label*]

*flags* is `NEW_LABEL`, `NO_CORRECTION`, or is 0 (zero). 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*.

`stobs1()` translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *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

When these functions return zero, `error` contains one of the following 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's *sensitivity label* and the process does not have `PRIV_SYS_TRANS_LABEL` in its set of effective privileges.

stobcl(3TSOL)

>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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), blvalid(3TSOL),  
hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

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.

stobclear(3TSOL)

NAME	stobl, stobcl, stobsl, stobclear – translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  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 stobclear(const char *string, bclear_t *clearance, const int flags,               int *error) ;</pre>				
DESCRIPTION	<p>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's sensitivity label.</p> <p>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.</p> <p>The generic form of an input character-coded label string is:</p> <pre>[ [ + ] classification name ] [ [ +   - ] word ... ]</pre> <p>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 classification name followed by a + or –, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels (see <code>hextob()</code>) when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.</td></tr> </table>	NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.	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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.
NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.				
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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.				



0 (zero)                      The default action is taken.

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

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

[*sensitivity label*]

*flags* is *NEW\_LABEL*, *NO\_CORRECTION*, or is 0 (zero). 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*.

*stobs1()* translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *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*.

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

When these functions return zero, *error* contains one of the following 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's *sensitivity label* and the process does not have *PRIV\_SYS\_TRANS\_LABEL* in its set of effective privileges.

stobclear(3TSOL)

>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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolor(3TSOL), blvalid(3TSOL),  
hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

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	stobl, stobcl, stobsl, stobclear – translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  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 stobclear(const char *string, bclear_t *clearance, const int flags,               int *error) ;</pre>				
DESCRIPTION	<p>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's sensitivity label.</p> <p>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.</p> <p>The generic form of an input character-coded label string is:</p> <pre>[ [ + ] classification name ] [ [ +   - ] word ... ]</pre> <p>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 classification name followed by a + or –, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels (see <code>hextob()</code>) when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.</td></tr> </table>	NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.	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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.
NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.				
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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.				

stobl(3TSOL)

0 (zero)

The default action is taken.

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

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

[*sensitivity label*]

*flags* is `NEW_LABEL`, `NO_CORRECTION`, or is 0 (zero). 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*.

`stobs1()` translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *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

When these functions return zero, `error` contains one of the following 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's *sensitivity label* and the process does not have `PRIV_SYS_TRANS_LABEL` in its set of effective privileges.

stobl(3TSOL)

>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 contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

bcltobanner(3TSOL), blcompare(3TSOL), bltocolour(3TSOL), blvalid(3TSOL),  
hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES**

attributes(5)

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.

stobsl(3TSOL)

NAME	stobl, stobcl, stobsl, stobclear – translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/label.h&gt;  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 stobclear(const char *string, bclear_t *clearance, const int flags,               int *error) ;</pre>				
DESCRIPTION	<p>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's sensitivity label.</p> <p>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.</p> <p>The generic form of an input character-coded label string is:</p> <pre>[ [ + ] classification name ] [ [ +   - ] word ... ]</pre> <p>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 classification name followed by a + or –, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels (see <code>hextob()</code>) when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.</td></tr> </table>	NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.	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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.
NEW_LABEL	<i>label</i> contents is not used, is formatted as a label of the relevant type, and is assumed to be ADMIN_LOW 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's sensitivity label.				
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 <code>label_encodings</code> file. The NO_CORRECTION flag implies the NEW_LABEL flag.				

0 (zero)                      The default action is taken.

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

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

[*sensitivity label*]

*flags* is `NEW_LABEL`, `NO_CORRECTION`, or is 0 (zero). 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*.

`stobsl()` translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *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

When these functions return zero, `error` contains one of the following 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's *sensitivity label* and the process does not have `PRIV_SYS_TRANS_LABEL` in its set of effective privileges.

stobsl(3TSOL)

**FILES** >0 The character-coded label *string* is in error. *error* is a one-based index into *string* indicating where the translation error occurred.

/etc/security/tsol/label\_encodings

The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual** bcltobanner(3TSOL), blcompare(3TSOL), bltocolor(3TSOL), blvalid(3TSOL),  
hextob(3TSOL)

*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual  
NOTES** attributes(5)

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.



str\_to\_auth(3TSOL)

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
SYNOPSIS	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
DESCRIPTION	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.

str\_to\_auth\_set(3TSOL)

<b>NAME</b>	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text – translate and verify user authorizations
<b>SYNOPSIS</b>	<b>cc</b> [ <i>flag...</i> ] <i>file...</i> -ltsol -ltsolddb -lcmd -lnsl [ <i>library...</i> ] <b>(obsolete)</b>
<b>DESCRIPTION</b>	These functions are obsolete. Authorizations in Trusted Solaris 8 and later releases do not need translation. See getauthattr(3SECDB) for how to search auth_attr(4) entries.

NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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.</p> <p>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 <code>file_mac_read,file_mac_write,none,proc_nofloat</code> produces the same result as <code>proc_nofloat</code> alone. The constructed privilege set is stored in the <i>priv_set_t</i> buffer to which <i>priv_set</i> points.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

str\_to\_priv(3TSOL)

**RETURN VALUES**

get_priv_text()	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
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 NULL. The function returns NULL 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 TSOL_MAX_PRIV.
str_to_priv_set()	Returns NULL 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.

**ERRORS**

priv_to_str() may fail for this reason:	
EINVAL	The specified <i>priv_id</i> is greater than TSOL_MAX_PRIV.
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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

priv\_desc(4) priv\_name(4)  
attributes(5)

str\_to\_priv\_set(3TSOL)

NAME	priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]  #include &lt;tsol/priv.h&gt;  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>						
DESCRIPTION	<p>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.</p> <p>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.</p> <p>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.</p> <p>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 <code>file_mac_read,file_mac_write,none,proc_nofloat</code> produces the same result as <code>proc_nofloat</code> alone. The constructed privilege set is stored in the <i>priv_set_t</i> buffer to which <i>priv_set</i> points.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

str\_to\_priv\_set(3TSOL)

RETURN VALUES	get_priv_text()	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
	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 NULL. The function returns NULL 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 TSOL_MAX_PRIV.
	str_to_priv_set()	Returns NULL 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.
ERRORS	priv_to_str() may fail for this reason:	
	EINVAL	The specified <i>priv_id</i> is greater than TSOL_MAX_PRIV.
	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.
NOTES	EFAULT	The specified <i>priv_name</i> is an invalid address.
	To use these routines, the program must be loaded with the Trusted Solaris library libtsol or libtsol.so.	
Trusted Solaris 8 4/01 Reference Manual	priv_desc(4)	priv_name(4)
	attributes(5)	

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 <code>static</code> 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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-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 <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

## svc\_auth\_reg(3NSL)

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

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

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

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

```
void xprt_register(const SVCXPRT *xprt);
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

```
void xprt_unregister(const SVCXPRT *xprt);
```

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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, `svc_unreg()` must be called with the privilege on.



	svc_auth_reg(3NSL)
	<p>The PRIV_NET_PRIVADDR privilege is required for <code>rpc_reg()</code>, <code>rpc_svc()</code>, or <code>svc_unreg()</code> calls that create or delete mappings for a transport that uses a privileged address.</p> <p>The PRIV_NET_SETID privilege is required by <code>svc_unreg()</code> in order for anyone other than the owner of a mapping to delete the mapping.</p>
Trusted Solaris 8 4/01 Reference Manual	inetd(1M), rpcbind(1M), rpc(3NSL), rpc_svc_calls(3NSL), rpc_svc_create(3NSL), rpcbind(3NSL)
SunOS 5.8 Reference Manual	select(3C), rpc_svc_err(3NSL), attributes(5)

## svc\_control(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td><code>SVCGET_XID</code></td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	<code>SVCGET_XID</code>	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
<code>SVCGET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.						
<code>SVCSET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
<code>SVCGET_XID</code>	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_control(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const
rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);
    svc_create() creates server handles for all the transports belonging to the class
    nettype.
```

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

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

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

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

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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

## svc\_control(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**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 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.

**Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_reg(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`  
  
`rpc_svc_err(3NSL)`, `attributes(5)`

## svc\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td><code>SVCGET_XID</code></td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	<code>SVCGET_XID</code>	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
<code>SVCGET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.						
<code>SVCSET_VERSQUIET</code>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
<code>SVCGET_XID</code>	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const
rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);
svc_create() creates server handles for all the transports belonging to the class
nettype.
```

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

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

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

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

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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

## svc\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe



	svc_create(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

svc\_destroy(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_destroy(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const
rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);
    svc_create() creates server handles for all the transports belonging to the class
    nettype.
```

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

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

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

```
SVCXPRT *svc_dg_create(const int fildes, const uint_t sendsz, const uint_t recvsz);
    This routine creates a connectionless RPC service handle, and returns a pointer to it.
    This routine returns NULL if it fails, and an error message is logged. 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(1M).
```

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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
```

## svc\_destroy(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

	svc_destroy(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

## svc\_dg\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_dg\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

```
int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const
rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);
    svc_create() creates server handles for all the transports belonging to the class
    nettype.
```

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

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

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

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

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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

## svc\_dg\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe



	svc_dg_create(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

## svc\_dg\_enablecache(3NSL)

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpirt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpirt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpirt);</pre> <p>This function frees resources allocated to service a client request directed to the service endpoint <i>xpirt</i>. This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> is invoked, the service endpoint <i>xpirt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_dg_enablecache(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_dg\_enablecache(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_dg\_enablecache(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xpirt\_register(3NSL), attributes(5)

#### 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_dg_enablecache(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

svc\_done(3NSL)

<b>NAME</b>	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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code>  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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_done(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.



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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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_getrpccaller(const SVCXPRT *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_done(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_done(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_exit(3NSL)

<b>NAME</b>	<code>rpc_svc_calls</code> , <code>svc_dg_enablecache</code> , <code>svc_done</code> , <code>svc_exit</code> , <code>svc_fdset</code> , <code>svc_freeargs</code> , <code>svc_getargs</code> , <code>svc_getreq_common</code> , <code>svc_getreq_poll</code> , <code>svc_getreqset</code> , <code>svc_getrpcaller</code> , <code>svc_max_pollfd</code> , <code>svc_pollfd</code> , <code>svc_run</code> , <code>svc_sendreply</code> – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpvt);</pre> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_exit(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_exit(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval) ;
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_exit(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xpirt\_register(3NSL), attributes(5)

#### 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_exit(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.



svc\_fd\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

## svc\_fd\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

*int* svc\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const char \**nettype*);  
*svc\_create()* creates server handles for all the transports belonging to the class *nettype*.

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

*void* svc\_destroy(SVCXPRT \**xprt*);

A function macro that 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 *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);  
This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 *fildev*, const uint\_t *sendsz*, const uint\_t *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

svc\_fd\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*))(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

svc\_fd\_create(3NSL)

**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 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.

**Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_reg(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`  
  
`rpc_svc_err(3NSL)`, `attributes(5)`

<b>NAME</b>	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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code>  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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_fdset(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_fdset(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_fdset(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_freeargs(3NSL)

<b>NAME</b>	<code>rpc_svc_calls</code> , <code>svc_dg_enablecache</code> , <code>svc_done</code> , <code>svc_exit</code> , <code>svc_fdset</code> , <code>svc_freeargs</code> , <code>svc_getargs</code> , <code>svc_getreq_common</code> , <code>svc_getreq_poll</code> , <code>svc_getreqset</code> , <code>svc_getrpcaller</code> , <code>svc_max_pollfd</code> , <code>svc_pollfd</code> , <code>svc_run</code> , <code>svc_sendreply</code> – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpvt);</pre> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_freeargs(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_freeargs(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval) ;
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_freeargs(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xpirt\_register(3NSL), attributes(5)

#### 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_freeargs(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

svc\_getargs(3NSL)

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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code> This function allocates a duplicate request cache for the service endpoint <code>xpvt</code>, large enough to hold <code>cache_size</code> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code> This function frees resources allocated to service a client request directed to the service endpoint <code>xpvt</code>. This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> is invoked, the service endpoint <code>xpvt</code> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_getargs(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.



## svc\_getargs(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_getargs(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_getargs(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_getreq\_common(3NSL)

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code></p> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code></p> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_getreq_common(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_getreq\_common(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval) ;
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_getreq\_common(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

**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(3SOCKET) 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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
  
**SunOS 5.8  
Reference Manual**

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xpirt\_register(3NSL), attributes(5)

**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(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.



svc\_getreq\_poll(3NSL)

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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code> This function allocates a duplicate request cache for the service endpoint <code>xpvt</code>, large enough to hold <code>cache_size</code> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code> This function frees resources allocated to service a client request directed to the service endpoint <code>xpvt</code>. This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> is invoked, the service endpoint <code>xpvt</code> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_getreq\_poll(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_getreq\_poll(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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_getrpccaller(const SVCXPRT *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_getreq\_poll(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_poll(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_getreqset(3NSL)

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpvt);</pre> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_getreqset(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_getreqset(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.



svc\_getreqset(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xpirt\_register(3NSL), attributes(5)

#### 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_getreqset(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code>  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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## `svc_getrpccaller(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_getrpcaller(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval) ;
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_getrpccaller(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_getrpccaller(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_max\_pollfd(3NSL)

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpvt);</pre> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>



`svc_max_pollfd(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_max\_pollfd(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_max\_pollfd(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

**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(3SOCKET) 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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
  
**SunOS 5.8  
Reference Manual**

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xpirt\_register(3NSL), attributes(5)

**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_max_pollfd(3NSL)`

`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_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xpvt);</code>  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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_pollfd(3NSL)

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_pollfd(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpvt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 *pfds, const int pollretval) ;
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *pfds* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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);
```

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, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_pollfd(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_pollfd(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_raw\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

`svc_raw_create(3NSL)`

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the `SVCGET_XID` request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type `SVCGET_XID`, `svc_control()` will return `FALSE`. Note also that the transaction ID read by the server can be set by the client through the suboption `CLSET_XID` in `clnt_control()`. See `clnt_create(3NSL)`

`int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`  
`svc_create()` creates server handles for all the transports belonging to the class *nettype*.

*nettype* defines a class of transports which 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 *nettype* is `NULL`, it defaults to `netpath`.

`svc_create()` registers itself with the `rpcbind` service [see `rpcbind(1M)`]. *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(3NSL)`). If `svc_create()` succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

`void svc_destroy(SVCXPRT *xpirt);`

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

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

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. *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(1M)`.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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

## svc\_raw\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

	svc_raw_create(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

## svc\_reg(3NSL)

<b>NAME</b>	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
<b>DESCRIPTION</b>	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-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 <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

svc\_reg(3NSL)

If the server has the PRIV\_NET\_MAC\_READ privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the PRIV\_NET\_PRIVADDR privilege.

The PRIV\_NET\_SETID privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

int svc\_auth\_reg(const int cred\_flavor, const enum auth\_stat (\*handler)());

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

void xprt\_register(const SVCXPRT \*xprt);

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

void xprt\_unregister(const SVCXPRT \*xprt);

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

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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, `svc_unreg()` must be called with the privilege on.

svc\_reg(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`



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_getrpccaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <p><code>int svc_dg_enablecache(SVCXPRT *xprt, const uint_t cache_size);</code>  This function allocates a duplicate request cache for the service endpoint <i>xprt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT *xprt);</code>  This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i>. This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

## svc\_run(3NSL)

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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

fd\_set svc\_fdset;

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

pollfd\_t \*svc\_pollfd;

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

int svc\_max\_pollfd;

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

bool\_t svc\_freeargs(const SVCXPRT \*xprt, const xdrproc\_t inproc, caddr\_t in);

A function macro that 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, and FALSE otherwise.

## svc\_run(3NSL)

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

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

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xpirt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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_getrpccaller(const SVCXPRT *xpirt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xpirt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

svc\_run(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xprt, const xdrproc\_t outproc, const caddr\_t out);

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

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

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

#### 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(3SOCKET) 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.

#### Trusted Solaris 8 4/01 Reference Manual

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL),  
t6alloc\_blk(3NSL), t6free\_blk(3NSL)

#### SunOS 5.8 Reference Manual

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C),  
xprt\_register(3NSL), attributes(5)

#### 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_run(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

## svc\_sendreply(3NSL)

<b>NAME</b>	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_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> 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 <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  int svc_dg_enablecache(SVCXPRT *xpvt, const uint_t cache_size);</pre> <p>This function allocates a duplicate request cache for the service endpoint <i>xpvt</i>, large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <pre>int svc_done(SVCXPRT *xpvt);</pre> <p>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 <code>User MT</code> mode. In the <code>User MT</code> 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 <code>svc_done()</code> 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 <code>rpc_control()</code> call.</p> <p>This function is safe in MT applications. It will have no effect if invoked in modes other than the <code>User MT</code> mode.</p>

`svc_sendreply(3NSL)`

`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(3NSL)` functions, or using `xprt_register(3NSL)`.

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

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest 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_getregset()` 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.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest 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_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3NSL)` to remove this limitation.

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

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

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

A function macro that 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, and FALSE otherwise.

## svc\_sendreply(3NSL)

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);
```

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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. It is assumed to be an array 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 only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It 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 *xprt);
```

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt*.

This function macro is safe in MT applications.

```
void svc_run(void);
```

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one 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, it will create threads to service client requests. In the `User` MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.



svc\_sendreply(3NSL)

bool\_t svc\_sendreply(const SVCXPRT \*xpirt, const xdrproc\_t outproc, const caddr\_t out);

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

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

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

**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(3SOCKET) 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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
  
**SunOS 5.8  
Reference Manual**

rpc(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), libt6(3NSL), t6alloc\_blk(3NSL), t6free\_blk(3NSL)

rpcgen(1), poll(2), rpc\_control(3NSL), rpc\_svc\_err(3NSL), select(3C), xpirt\_register(3NSL), attributes(5)

**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_sendreply(3NSL)`

`svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

svc\_tli\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_tli\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

*int* svc\_create(const void (\**dispatch*))(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const char \**nettype*);  
*svc\_create()* creates server handles for all the transports belonging to the class *nettype*.

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void svc\_destroy(SVCXPRT \**xprt*);

A function macro that 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 *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);  
This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 *fildev*, const uint\_t *sendsz*, const uint\_t *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

svc\_tli\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*))(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

svc\_tli\_create(3NSL)

**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 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.

**Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_reg(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`  
  
`rpc_svc_err(3NSL)`, `attributes(5)`

svc\_tp\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

## svc\_tp\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

`int svc_create(const void (*dispatch)(const struct svc_req *, const SVCXPRT *), const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);`  
*svc\_create()* creates server handles for all the transports belonging to the class *nettype*.

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

`void svc_destroy(SVCXPRT *xpirt);`

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

`SVCXPRT *svc_dg_create(const int fildes, const uint_t sendsz, const uint_t recvsz);`  
This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint_t sendsz, const uint_t 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



svc\_tp\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

**ATTRIBUTES** See attributes (5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

svc\_tp\_create(3NSL)

**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 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.

**Trusted Solaris 8  
4/01 Reference  
Manual  
  
SunOS 5.8  
Reference Manual**

`rpcbind(1M)`, `rpc(3NSL)`, `rpc_clnt_create(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_reg(3NSL)`, `libt6(3NSL)`, `t6alloc_blk(3NSL)`, `t6free_blk(3NSL)`  
  
`rpc_svc_err(3NSL)`, `attributes(5)`

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 <code>static</code> 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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-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 <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

## svc\_unreg(3NSL)

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

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

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

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

```
void xprt_register(const SVCXPRT *xprt);
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

```
void xprt_unregister(const SVCXPRT *xprt);
```

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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, `svc_unreg()` must be called with the privilege on.

svc\_unreg(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`

## svc\_vc\_create(3NSL)

<b>NAME</b>	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						
<b>DESCRIPTION</b>	<p>These routines are part of the RPC library which 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 <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>						
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;</pre> <pre>bool_t svc_control (SVCXPRT *svc, const uint_t req, void *info);</pre> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td>SVCGET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td>SVCSET_VERSQUIET</td><td>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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> <tr> <td>SVCGET_XID</td><td>Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program</td></tr> </table>	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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.	SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program
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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> 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 <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.						
SVCGET_XID	Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program						

svc\_vc\_create(3NSL)

number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET\_XID request, *\*info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET\_XID, *svc\_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET\_XID in *clnt\_control()*. See *clnt\_create(3NSL)*

*int* svc\_create(const void (\**dispatch*))(const struct svc\_req \*, const SVCXPRT \*), const  
rpcprog\_t *prognum*, const rpcvers\_t *versnum*, const char \**nettype*);  
*svc\_create()* creates server handles for all the transports belonging to the class *nettype*.

*nettype* defines a class of transports which 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 *nettype* is NULL, it defaults to netpath.

*svc\_create()* registers itself with the rpcbind service [see *rpcbind(1M)*]. *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(3NSL)*). If *svc\_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

*void* svc\_destroy(SVCXPRT \**xprt*);  
A function macro that 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 uint\_t *sendsz*, const uint\_t *recvsz*);  
This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. *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(1M)*.

Warning: since connectionless-based RPC messages can only hold 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 uint\_t *sendsz*, const uint\_t *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

## svc\_vc\_create(3NSL)

receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.

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(3NSL)`). 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 an error message is logged.

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 uint\_t *sendsz*, const uint\_t *recvsz*);

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 non-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 where 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 *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT \*svc\_tp\_create(const void (\**dispatch*)(const struct svc\_req \*, const SVCXPRT \*), const rpcprog\_t *prognum*, const rpcvers\_t *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 a NULL is returned and an error message is logged.

SVCXPRT \*svc\_vc\_create(const int *fildev*, const uint\_t *sendsz*, const uint\_t *recvsz*);

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

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe



	svc_vc_create(3NSL)
<b>SUMMARY OF TRUSTED SOLARIS CHANGES</b>	<p>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 <code>svc_create()</code> binds to a transport, a multilevel port will be created.</p> <p>Most <code>rpcbind()</code> services operate only on mappings that either match the sensitivity label of the server or are multilevel.</p> <p>The PRIV_NET_MAC_READ privilege affects the operation of several <code>rpcbind()</code> services. If the privilege is on when a library routine calls <code>rpcbind()</code> to create a mapping, a multilevel mapping is created.</p> <p>The PRIV_NET_PRIVADDR privilege is required when a library routine calls <code>rpcbind()</code> to create a mapping for a transport that uses a privileged address.</p> <p>The SVCXPRT structure allows a server to provide <code>t6attr_t</code> 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 <code>t6alloc_blk()</code> routine to allocate attribute-control structures and set the <code>t6attr_t</code> pointers in the SVCXPRT structure. When <code>svc_destroy()</code> is used to destroy a service handle, the server should also use <code>t6free_blk()</code> to free any attribute-control structures previously allocated for that service handle.</p>
<b>Trusted Solaris 8 4/01 Reference Manual</b>	<p><code>rpcbind(1M)</code>, <code>rpc(3NSL)</code>, <code>rpc_clnt_create(3NSL)</code>, <code>rpc_svc_calls(3NSL)</code>, <code>rpc_svc_reg(3NSL)</code>, <code>libt6(3NSL)</code>, <code>t6alloc_blk(3NSL)</code>, <code>t6free_blk(3NSL)</code></p>
<b>SunOS 5.8 Reference Manual</b>	<p><code>rpc_svc_err(3NSL)</code>, <code>attributes(5)</code></p>

t6alloc\_blk(3NSL)

NAME	t6alloc_blk, t6free_blk – Allocate and free security-attribute control structure and buffer						
SYNOPSIS	<pre>cc [flags...] file ... -lt6  #include &lt;tsix/t6attrs.h&gt;  t6attr_t t6alloc_blk(t6mask_t mask);  void t6free_blk(t6attr_t t6ctl);</pre>						
DESCRIPTION	<p>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(3NSL) and t6set_attr(3NSL) 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 mask parameter.</p> <p>t6free_blk() should be used in conjunction with t6alloc_blk() to free the opaque control structure and any space within it.</p>						
RETURN VALUES	Upon successful completion, t6alloc_blk() returns a pointer to the t6attr_t structure. Upon failure, t6alloc_blk() returns a NULL pointer.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual Warnings	<p>libt6(3NSL), t6get_attr(3NSL), t6set_attr(3NSL)</p> <p>attributes(5)</p> <p>For generic TSIX applications, use t6free_blk() to free memory allocated by t6alloc_blk() for better portability.</p>						
NOTES	<p>This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; this interface is available in TSIX(RE) 1.1-API-compliant systems.</p>						

<b>NAME</b>	t6attr_query – Get mask indicating which attributes came from templates
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt; int t6attr_query(int fd , t6mask_t *mask);</pre>
<b>DESCRIPTION</b>	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.
<b>RETURN VALUES</b>	t6attr_query() returns: <ul style="list-style-type: none"> <li>0            On success.</li> <li>-1          If an error is encountered.</li> </ul>
<b>NOTES</b>	This interface is specific to the Trusted Solaris environment.

t6clear\_blk(3NSL)

NAME	t6clear_blk – Clear security attributes
SYNOPSIS	<pre>cc [flags...] file ... -lt6  #include &lt;tsix/t6attrs.h&gt;  void t6clear_blk(t6mask_t mask, t6attr_t src);</pre>
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.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

Trusted Solaris 8  
4/01 Reference  
Manual  
Reference Manual  
NOTES

libt6(3NSL), t6alloc\_blk(3NSL)  
attributes(5)  
This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; 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 &lt;tsix/t6attrs.h&gt; int t6cmp_blk(const t6attr_t ctl1, const t6attr_t ctl2);</pre>						
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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	This call returns zero if <i>ctl1</i> equals <i>ctl2</i> ; otherwise, the call returns a nonzero value.						
Trusted Solaris 8 4/01 Reference Manual SunOS 5.6 Reference Manual	libt6(3NSL), t6alloc_blk(3NSL) attributes(5)						
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.						

t6copy\_blk(3NSL)

NAME	t6copy_blk – copy security attributes						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt;  int t6copy_blk(const t6attr_t attr1, t6attr_t attr2);</pre>						
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(3NSL) for more details.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	t6copy_blk() returns: 0           On success. -1           On failure. This call will fail if an attribute in <i>attr1</i> is not allocated in <i>attr2</i> . Nothing is copied for this failure case.						
Trusted Solaris 8 4/01 Reference Manual NOTES	libt6(3NSL), t6alloc_blk(3NSL) attributes(5)  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.						

<b>NAME</b>	t6dup_blk – Duplicate security attributes						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt; t6attr_t t6dup_blk(const t6attr_t src);</pre>						
<b>DESCRIPTION</b>	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.						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>RETURN VALUES</b>	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.						
<b>DIAGNOSTICS</b>	ENOMEM                      Out of memory for allocation						
<b>References</b>	libt6(3NSL), t6alloc_blk(3NSL)						
<b>See also</b>	attributes(5)						
<b>NOTES</b>	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.						

t6ext\_attr(3NSL)

NAME	t6ext_attr, t6new_attr – manipulate network-endpoint security options						
SYNOPSIS	<pre>cc [flags...] file ... -lt6  #include &lt;tsix/t6attrs.h&gt;  int t6ext_attr(int fd, t6cmd_t cmd);  int t6new_attr(int fd, t6cmd_t cmd);</pre>						
DESCRIPTION	<p>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().</p> <p>t6new_attr() with a <i>cmd</i> 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(3NSL) returns valid security attributes only when a change in the attributes is detected. This situation is indicated by setting the t6recvfrom() parameter <i>*new_mask</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().</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>libt6(3NSL), t6recvfrom(3NSL)</p> <p>attributes(5)</p> <p>In the Trusted Solaris environment, t6ext_attr() is a NULL function.</p> <p>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.</p>						



<b>NAME</b>	t6alloc_blk, t6free_blk – Allocate and free security-attribute control structure and buffer						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt;  t6attr_t t6alloc_blk(t6mask_t mask); void t6free_blk(t6attr_t t6ctl);</pre>						
<b>DESCRIPTION</b>	<p>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(3NSL) and t6set_attr(3NSL) 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 mask parameter.</p> <p>t6free_blk() should be used in conjunction with t6alloc_blk() to free the opaque control structure and any space within it.</p>						
<b>RETURN VALUES</b>	Upon successful completion, t6alloc_blk() returns a pointer to the t6attr_t structure. Upon failure, t6alloc_blk() returns a NULL pointer.						
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	libt6(3NSL), t6get_attr(3NSL), t6set_attr(3NSL)						
<b>WARNINGS</b>	<p>attributes(5)</p> <p>For generic TSIX applications, use t6free_blk() to free memory allocated by t6alloc_blk() for better portability.</p>						
<b>NOTES</b>	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; this interface is available in TSIX(RE) 1.1-API-compliant systems.						

t6get\_attr(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>								
DESCRIPTION	<p>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 <i>t6ctl</i>. <i>attr_type</i> contains a number (defined in &lt;tsix/t6attrs.h&gt;) that specifies which type of attribute the caller is interested in getting. Only one type can be specified per call.</p> <p>Returned value by t6get_attr() should be type cast to the standard type that represents the type indicated by <i>attr_type</i>.</p> <p>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 &lt;tsix/t6attrs.h&gt;.</p>								
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.								
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:								
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtsu								
MT-Level	MT-Safe								
Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual	libt6(3NSL), t6alloc_blk(3NSL), t6free_blk(3NSL)								
NOTES	<p>attributes(5)</p> <p>In the Trusted Solaris environment, t6get_attr() returns values of these types:</p> <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>bslabel_t</td><td>Sensitivity label</td></tr></table>	au_id_t	Audit ID	auditinfo_t	Audit info	bclear_t	Clearance	bslabel_t	Sensitivity label
au_id_t	Audit ID								
auditinfo_t	Audit info								
bclear_t	Clearance								
bslabel_t	Sensitivity label								

t6get\_attr(3NSL)

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.

## t6get\_endpt\_default(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>
DESCRIPTION	<p>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' effective attributes.</p> <p>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.</p> <p>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>.</p> <p>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 the attribute is taken from the endpoint's default attributes.</p> <p>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(3NSL).</p> <p>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(3NSL).</p> <p>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.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

t6get\_endpt\_default(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**RETURN VALUES** Upon successful completion, these calls return 0. If either call encounters an error, the call returns -1.

**Trusted Solaris 8** libt6(3NSL), t6sendto(3NSL), t6set\_attr(3NSL)  
**4/01 Reference**  
**SunOS 5.10** attributes(5)  
**Reference Manual**

**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.

## t6get\_endpt\_mask(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>
DESCRIPTION	<p>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' effective attributes.</p> <p>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.</p> <p>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>.</p> <p>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 the attribute is taken from the endpoint's default attributes.</p> <p>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(3NSL).</p> <p>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(3NSL).</p> <p>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.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

t6get\_endpt\_mask(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**RETURN VALUES** Upon successful completion, these calls return 0. If either call encounters an error, the call returns -1.

**Trusted Solaris 8** libt6(3NSL), t6sendto(3NSL), t6set\_attr(3NSL)  
**4/01 Reference**  
**SunOS 5.10** attributes(5)  
**Reference Manual**

**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.

t6last\_attr(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>						
DESCRIPTION	<p>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().</p> <p>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.</p> <p>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(2)), in which case -1 is returned with the external variable errno set to EWOULDBLOCK.</p> <p>In order to obtain audit ID, additional audit information, and supplementary group IDs when using connectionless transports, connect(3SOCKET) must be used to associate the peer network address with the local transport endpoint before calling t6peek_attr() or t6last_attr().</p>						
RETURN VALUES	Upon successful completion, these calls return 0, place the retrieved security attributes in the t6attr_t structure, and set *new_attrs 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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual SunOS 5.9 Reference Manual NOTES	<p>libt6(3NSL), fcntl(2)</p> <p>connect(3SOCKET), attributes(5)</p> <p>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.</p>						



<b>NAME</b>	t6ext_attr, t6new_attr – manipulate network-endpoint security options						
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt;  int t6ext_attr(int fd, t6cmd_t cmd); int t6new_attr(int fd, t6cmd_t cmd);</pre>						
<b>DESCRIPTION</b>	<p>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().</p> <p>t6new_attr() with a <i>cmd</i> 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(3NSL) returns valid security attributes only when a change in the attributes is detected. This situation is indicated by setting the t6recvfrom() parameter <i>*new_mask</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().</p>						
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
<b>Trusted Solaris 8 4/01 Reference Manual</b>	libt6(3NSL), t6recvfrom(3NSL)						
<b>NOTES</b>	<p>attributes(5)</p> <p>In the Trusted Solaris environment, t6ext_attr() is a NULL function.</p> <p>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.</p>						

t6peek\_attr(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>						
DESCRIPTION	<p>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().</p> <p>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.</p> <p>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(2)), in which case -1 is returned with the external variable errno set to EWOULDBLOCK.</p> <p>In order to obtain audit ID, additional audit information, and supplementary group IDs when using connectionless transports, connect(3SOCKET) must be used to associate the peer network address with the local transport endpoint before calling t6peek_attr() or t6last_attr().</p>						
RETURN VALUES	Upon successful completion, these calls return 0, place the retrieved security attributes in the t6attr_t structure, and set *new_attrs 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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>libt6(3NSL), fcntl(2)</p> <p>connect(3SOCKET), attributes(5)</p> <p>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.</p>						

NAME	t6recvfrom – read security attributes and data from a trusted endpoint				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl -lt6 [library...]  #include &lt;tsix/t6attrs.h&gt;  ssize_t t6recvfrom(int sock, void *buffer, size_t len, int flags, struct     sockaddr *name, Psocklen_t namelenp, t6attr_t handle, t6mask_t     *new_mask) ;</pre>				
DESCRIPTION	<p>t6recvfrom() receives data and its associated security attributes from a communication endpoint. The <i>name</i> and <i>namelenp</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>name</i> is not a NULL pointer, the source address of the message is filled in. <i>namelenp</i> is a value-result parameter, initialized to the size of the buffer associated with <i>name</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(3SOCKET).)</p> <p>The <i>flags</i> parameter is formed by ORing one or more of these values:</p> <table> <tr> <td>MSG_OOB</td><td>Read any out-of-band data present on the socket rather than the regular in-band data. If <i>handle</i> is not NULL, out-of-band data security attributes are also retrieved.</td></tr> <tr> <td>MSG_PEEK</td><td>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>handle</i> is not NULL, security attributes of the data are also peeked.</td></tr> </table> <p><i>handle</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(3NSL). 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.</p> <p>If the t6new_attr(3NSL) 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_mask</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.</p> <p>In order to obtain audit ID, additional audit information, and supplementary group IDs when using connectionless transports, connect(3SOCKET) must be used to associate the peer network address with the local transport endpoint before calling t6recvfrom().</p>	MSG_OOB	Read any out-of-band data present on the socket rather than the regular in-band data. If <i>handle</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>handle</i> is not NULL, security attributes of the data are also peeked.
MSG_OOB	Read any out-of-band data present on the socket rather than the regular in-band data. If <i>handle</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>handle</i> is not NULL, security attributes of the data are also peeked.				
RETURN VALUES	Upon success, t6recvfrom() returns the number of bytes read. Upon failure, t6recvfrom() returns -1 and sets errno.				

t6recvfrom(3NSL)

## 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	<i>sock</i> 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	<i>sock</i> is not a socket.
ESTALE	A stale NFS file handle exists.

## ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

## Trusted Solaris 8 4/01 Reference Manual NOTES

`libt6(3NSL)`, `t6get_attr(3NSL)`, `t6sendto(3NSL)`, `socket(3SOCKET)`  
`connect(3SOCKET)`, `attributes(5)`

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 [flags...] file ... -lsocket -lnsl -lt6 [library...]  #include &lt;tsix/t6attrs.h&gt;  ssize_t t6sendto(int sock, const char *msg, size_t len, int flags,     const struct sockaddr *name, socklen_t namelen, const t6attr_t     handle) ;</pre>				
DESCRIPTION	<p>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.</p> <p><i>sock</i> is a socket created with socket(3SOCKET). The address of the target is given by <i>name</i> with <i>namelen</i> specifying its size. The length of the message is given by <i>len</i>.</p> <p>The <i>name</i> pointer and <i>namelen</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.</p> <p>A return value of -1 indicates locally detected errors only, not implicitly that the message was not delivered.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of these values:</p> <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> <p>The security attributes are specified by the <i>handle</i> parameter. To set up <i>handle</i>, see t6set_attr(3NSL).</p> <p>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_SETCLR, PRIV_NET_SETID, PRIV_NET_SETPRIV, PRIV_NET_BROADCAST.</p>	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.				

t6sendto(3NSL)

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**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:

<code>EBADF</code>	<code>sock</code> is an invalid file descriptor.
<code>EDESTADDRREQ</code>	A destination address is not specified.
<code>EINTR</code>	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
<code>EINVAL</code>	<code>namelen</code> is not the size of a valid address for the specified address family.
<code>EMSGSIZE</code>	The socket requires that message be sent atomically, and the message was too long.
<code>ENOMEM</code>	There was insufficient memory available to complete the operation.
<code>ENOSR</code>	There were insufficient STREAMS resources available for the operation to complete.
<code>ENOTSOCK</code>	<code>sock</code> is not a socket.

**Trusted Solaris 8  
4/01 Reference  
Manual**

`libt6(3NSL)`, `t6set_attr(3NSL)`, `t6set_endpt_default(3NSL)`,  
`socket(3SOCKET)`, *Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

`attributes(5)`

**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.

<b>NAME</b>	t6get_attr, t6set_attr – get security attributes from or set security attributes in the security-attribute buffer handled by a control structure								
<b>SYNOPSIS</b>	<pre>cc [flags...] file ... -lt6  #include &lt;tsix/t6attrs.h&gt;  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>								
<b>DESCRIPTION</b>	<p>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 <i>t6ctl</i>. <i>attr_type</i> contains a number (defined in &lt;tsix/t6attrs.h&gt;) that specifies which type of attribute the caller is interested in getting. Only one type can be specified per call.</p> <p>Returned value by t6get_attr() should be type cast to the standard type that represents the type indicated by <i>attr_type</i>.</p> <p>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 &lt;tsix/t6attrs.h&gt;.</p>								
<b>RETURN VALUES</b>	<p>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.</p>								
<b>ATTRIBUTES</b>	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtsu								
MT-Level	MT-Safe								
<b>Trusted Solaris 8 4/01 Reference Manual</b>	libt6(3NSL), t6alloc_blk(3NSL), t6free_blk(3NSL)								
<b>NOTES</b>	<p>attributes(5)</p> <p>In the Trusted Solaris environment, t6get_attr() returns values of these types:</p> <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>bslabel_t</td><td>Sensitivity label</td></tr> </table>	au_id_t	Audit ID	auditinfo_t	Audit info	bclear_t	Clearance	bslabel_t	Sensitivity label
au_id_t	Audit ID								
auditinfo_t	Audit info								
bclear_t	Clearance								
bslabel_t	Sensitivity label								

t6set\_attr(3NSL)

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.



t6set\_endpt\_default(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>
DESCRIPTION	<p>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' effective attributes.</p> <p>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.</p> <p>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>.</p> <p>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 the attribute is taken from the endpoint's default attributes.</p> <p>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(3NSL).</p> <p>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(3NSL).</p> <p>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.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

t6set\_endpt\_default(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

#### RETURN VALUES

Upon successful completion, these calls return 0. If either call encounters an error, the call returns -1.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**NOTES**

libt6(3NSL), t6sendto(3NSL), t6set\_attr(3NSL)

attributes(5)

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.

t6set\_endpt\_mask(3NSL)

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 &lt;tsix/t6attrs.h&gt;  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>
DESCRIPTION	<p>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' effective attributes.</p> <p>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.</p> <p>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>.</p> <p>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 the attribute is taken from the endpoint's default attributes.</p> <p>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(3NSL).</p> <p>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(3NSL).</p> <p>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.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

t6set\_endpt\_mask(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

#### RETURN VALUES

Upon successful completion, these calls return 0. If either call encounters an error, the call returns -1.

**Trusted Solaris 8**  
**4/01 Reference**  
**Manual**  
**NOTES**

libt6(3NSL), t6sendto(3NSL), t6set\_attr(3NSL)

attributes(5)

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	t6size_attr – Get the size of a particular attribute from the control structure						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include &lt;tsix/t6attrs.h&gt; int t6size_attr(t6attr_id_t attr_type, const t6attr_t t6ctl);</pre>						
RETURN VALUES	<p>t6size_attr() returns the size of an attribute indicated by <i>attr_type</i>.</p> <p>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 attr_type is invalid, t6size_attr() returns 0.</p> <p>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 attr_type is invalid or not in the <i>t6ctl</i>, t6size_attr() returns 0.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtsu</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
Trusted Solaris 8 4/01 Reference Manual NOTES	<p>t6copy_blk(3NSL), t6dup_blk(3NSL)</p> <p>attributes(5)</p> <p>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.</p>						

## t\_accept(3NSL)

NAME	t_accept – Accept a connection request
SYNOPSIS	<pre>#include &lt;xti.h&gt;  int t_accept(int fd, int resfd, const struct t_call *call);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, &lt;tiuser.h&gt;, must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is issued by a transport user to accept a connection request. The parameter <i>fd</i> identifies the local transport endpoint where the connection 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. The parameter <i>call</i> points to a <i>t_call</i> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; In <i>call</i>, <i>addr</i> is the protocol address of the calling transport user, <i>opt</i> indicates any options associated with the connection, <i>udata</i> points to any user data to be returned to the caller, and <i>sequence</i> is the value returned by <i>t_listen</i>(3NSL) that uniquely associates the response with a previously received connection indication. The address of the caller, <i>addr</i> may be null (length zero). Where <i>addr</i> is not null then it may optionally be checked by XTI.</pre> <p>A transport user may accept a connection on either the same, or on a different, local transport endpoint than the one on which the connection indication arrived. Before the connection can be accepted on the same endpoint (<i>resfd</i>==<i>fd</i>), the user must have responded to any previous connection indications received on that transport endpoint by means of <i>t_accept</i>() or <i>t_snddis</i>(3NSL). Otherwise, <i>t_accept</i>() will fail and set <i>t_errno</i> to TINDOUT.</p> <p>If a different transport endpoint is specified (<i>resfd</i>!=<i>fd</i>), then the user may or may not choose to bind the endpoint before the <i>t_accept</i>() is issued. If the endpoint is not bound prior to the <i>t_accept</i>(), the endpoint must be in the T_UNBND state before the <i>t_accept</i>() is issued, and the transport provider will automatically bind it to an address that is appropriate for the protocol concerned. If the transport user chooses to bind the endpoint it must be bound to a protocol address with a <i>qlen</i> of zero and must be in the T_IDLE state before the <i>t_accept</i>() is issued.</p> <p>Responding endpoints should be supplied to <i>t_accept</i>() in the state T_UNBND.</p> <p>The call to <i>t_accept</i>() may fail with <i>t_errno</i> set to TLOOK if there are indications (for example connect or disconnect) waiting to be received on endpoint <i>fd</i>. Applications should be prepared for such a failure.</p>

t\_accept(3NSL)

The *udata* argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of t\_open(3NSL) or t\_getinfo(3NSL). If the *len* field of *udata* is zero, no data will be sent to the caller. All the *maxlen* fields are meaningless.

When the user does not indicate any option (*call*→*opt.len* = 0) the connection shall be accepted with the option values currently set for the responding endpoint *resfd*.

## RETURN VALUES

t\_accept() returns:

- 0            On success.
- 1           On failure, and sets *errno* to indicate the error.

## VALID STATES

*fd*: T\_INCON  
*resfd* (*fd*!=*resfd*): T\_IDLE, T\_UNBND

## ERRORS

On failure, *t\_errno* is set to one of the following:

TACCES	The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.
TBADADDR	The specified protocol address was in an incorrect format or contained illegal information.
TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.
TBADF	The file descriptor <i>fd</i> or <i>resfd</i> does not refer to a transport endpoint.
TBALOPT	The specified options were in an incorrect format or contained illegal information.
TBADSEQ	Either an invalid sequence number was specified, or a valid sequence number was specified but the connection request was aborted by the peer. In the latter case, its T_DISCONNECT event will be received on the listening endpoint.
TINDOUT	The function was called with <i>fd</i> == <i>resfd</i> but there are outstanding connection indications on the endpoint. Those other connection indications must be handled either by rejecting them by means of t_snddis(3NSL) or accepting them on a different endpoint by means of t_accept().
TLOOK	An asynchronous event has occurred on the transport endpoint referenced by <i>fd</i> and requires immediate attention.

## t\_accept(3NSL)

	TNOTSUPPORT	This function is not supported by the underlying transport provider.
	TOUTSTATE	The communications endpoint referenced by <i>fd</i> or <i>resfd</i> is not in one of the states in which a call to this function is valid.
	TPROTO	This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error ( <i>t_errno</i> ).
	TPROVMISMATCH	The file descriptors <i>fd</i> and <i>resfd</i> do not refer to the same transport provider.
	TRESADDR	This transport provider requires both <i>fd</i> and <i>resfd</i> to be bound to the same address. This error results if they are not.
	TRESQLEN	The endpoint referenced by <i>resfd</i> (where <i>resfd</i> != <i>fd</i> ) was bound to a protocol address with a <i>qlen</i> that is greater than zero.
	TSYSERR	A system error has occurred during execution of this function.
<b>TLI COMPATIBILITY</b>	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.	
<b>Interface Header</b>	The XTI interfaces use the header file, <i>&lt;xti.h&gt;</i> . TLI interfaces should <i>not</i> use this header. They should use the header:  <code>#include &lt;tiuser.h&gt;</code>	
<b>Error Description Values</b>	The <i>t_errno</i> values that can be set by the XTI interface and cannot be set by the TLI interface are:  TPROTO TINDOUT TPROVMISMATCH TRESADDR TRESQLEN	
<b>Option Buffer</b>	The format of the options in an <i>opt</i> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not specify the buffer format.  For more information refer to the <i>Transport Interfaces Programming Guide</i>	
<b>ATTRIBUTES</b>	See <i>attributes(5)</i> for descriptions of the following attributes:	



t\_accept(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**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(3SOCKET)` for more information.

**Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual**

`bind(3SOCKET)`, `t_optmgt(3NSL)`, `t_bind(3NSL)`

`t_connect(3NSL)`, `t_getinfo(3NSL)`, `t_getstate(3NSL)`, `t_listen(3NSL)`,  
`t_open(3NSL)`, `t_rcvconnect(3NSL)`, `t_snddis(3NSL)`, `attributes(5)`

*Transport Interfaces Programming Guide*

**WARNINGS**

There may be transport provider-specific restrictions on address binding.

Some transport providers do not differentiate between a connection indication and the connection itself. If the connection has already been established after a successful return of `t_listen(3NSL)`, `t_accept()` will assign the existing connection to the transport endpoint specified by *resfd*.

## t\_bind(3NSL)

NAME	t_bind – Bind an address to a transport endpoint
SYNOPSIS	<pre>#include &lt;xti.h&gt;  int t_bind(int fd, const struct t_bind *req, struct t_bind *ret);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the &lt;tiuser.h&gt; header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function associates a protocol address with the transport endpoint specified by <i>fd</i> and activates that transport endpoint. In connection mode, the transport provider may begin enqueueing incoming connect indications, or servicing a connection request on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint.</p> <p>The <i>req</i> and <i>ret</i> arguments point to a t_bind structure containing the following members:</p> <pre>struct netbuf    addr; unsigned         qlen;</pre> <p>The <i>addr</i> field of the t_bind structure specifies a protocol address, and the <i>qlen</i> field is used to indicate the maximum number of outstanding connection indications.</p> <p>The parameter <i>req</i> is used to request that an address, represented by the netbuf structure, be bound to the given transport endpoint. The parameter <i>len</i> specifies the number of bytes in the address, and <i>buf</i> points to the address buffer. The parameter <i>maxlen</i> has no meaning for the <i>req</i> argument. On return, <i>ret</i> contains an encoding for the address that the transport provider actually bound to the transport endpoint; if an address was specified in <i>req</i>, this will be an encoding of the same address. In <i>ret</i>, the user specifies <i>maxlen</i>, which is the maximum size of the address buffer, and <i>buf</i> which points to the buffer where the address is to be placed. On return, <i>len</i> specifies the number of bytes in the bound address, and <i>buf</i> points to the bound address. If <i>maxlen</i> equals zero, no address is returned. If <i>maxlen</i> is greater than zero and less than the length of the address, t_bind() fails with t_errno set to TBUFOVFLW.</p> <p>If the requested address is not available, t_bind() will return -1 with t_errno set as appropriate. If no address is specified in <i>req</i> (the <i>len</i> field of <i>addr</i> in <i>req</i> is zero or <i>req</i> is NULL), the transport provider will assign an appropriate address to be bound, and will return that address in the <i>addr</i> field of <i>ret</i>. If the transport provider could not allocate an address, t_bind() will fail with t_errno set to TNOADDR.</p> <p>The parameter <i>req</i> may be a null pointer if the user does not wish to specify an address to be bound. Here, the value of <i>qlen</i> is assumed to be zero, and the transport provider will assign an address to the transport endpoint. Similarly, <i>ret</i> may be a null pointer if the user does not care what address was bound by the provider and is not interested</p>

in the negotiated value of *qlen*. It is valid to set *req* and *ret* to the null pointer for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The *qlen* field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connection indications that the transport provider should support for the given transport endpoint. An outstanding connection indication is one that has been passed to the transport user by the transport provider but which has not been accepted or rejected. A value of *qlen* greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of *qlen* will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connection indications. However, this value of *qlen* will never be negotiated from a requested value greater than zero to zero. This is a requirement on transport providers; see WARNINGS below. On return, the *qlen* field in *ret* will contain the negotiated value.

If *fd* refers to a connection-mode service, this function allows more than one transport endpoint to be bound to the same protocol address. But it is not possible to bind more than one protocol address to the same transport endpoint. However, the transport provider must also support this capability. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connection indications associated with that protocol address. In other words, only one `t_bind()` for a given protocol address may specify a value of *qlen* greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connection indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of *qlen* greater than zero, `t_bind()` will return -1 and set `t_errno` to `TADDRBUSY`. When a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of the connection, until a `t_unbind(3NSL)` or `t_close(3NSL)` call has been issued. No other transport endpoints may be bound for listening on that same protocol address while that initial listening endpoint is active (in the data transfer phase or in the `T_IDLE` state). This will prevent more than one transport endpoint bound to the same protocol address from accepting connection indications.

If *fd* refers to connectionless mode service, this function allows for more than one transport endpoint to be associated with a protocol address, where the underlying transport provider supports this capability (often in conjunction with value of a protocol-specific option). If a user attempts to bind a second transport endpoint to an already bound protocol address when such capability is not supported for a transport provider, `t_bind()` will return -1 and set `t_errno` to `TADDRBUSY`.

**RETURN VALUES**

`t_bind()` returns:

- 0           On success.
- 1          On failure, and sets `t_errno` to indicate the error.

**VALID STATES**

`T_UNBND`

t\_bind(3NSL)

<b>ERRORS</b>	On failure, t_errno is set to one of the following:
	TACCES                    The user does not have permission to use the specified address.
	TADDRBUSY                The requested address is in use.
	TBADADDR                The specified protocol address was in an incorrect format or contained illegal information.
	TBADF                    The specified file descriptor does not refer to a transport endpoint.
	TBUFOVFLW                The number of bytes allowed for an incoming argument ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the value of that argument. The provider's state will change to T_IDLE and the information to be returned in <i>ret</i> will be discarded.
	TOUTSTATE                The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.
	TNOADDR                  The transport provider could not allocate an address.
	TPROTO                   This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).
	TSYSERR                  A system error has occurred during execution of this function.
<b>TLI COMPATIBILITY</b>	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.
<b>Interface Header</b>	The XTI interfaces use the header file, <xti.h>. TLI interfaces should <i>not</i> use this header. They should use the header:
	#include <tiuser.h>
<b>Address Bound</b>	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 than that requested.
<b>Error Description Values</b>	The t_errno values TPROTO and TADDRBUSY can be set by the XTI interface but cannot be set by the TLI interface.
	A t_errno value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the maxlen field of the corresponding buffer has been set to zero.
	For more information refer to the <i>Transport Interfaces Programming Guide</i>
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

t\_bind(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

## SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8  
4/01 Reference  
Manual

If the calling process possesses the `PRIV_NET_MAC_READ` privilege, the socket is bound to a multilevel port (MLP); otherwise, the connection is bound to a single-level port (SLP). See `bind(3SOCKET)` for more information.

`bind(3SOCKET)`, `t_accept(3NSL)`

`t_alloc(3NSL)`, `t_close(3NSL)`, `t_connect(3NSL)`, `t_unbind(3NSL)`,  
`attributes(5)`

## WARNINGS

The requirement that the value of *qlen* never be negotiated from a requested value greater than zero to zero implies that transport providers, rather than the XTI implementation itself, accept this restriction.

An implementation need not allow an application explicitly to bind more than one communications endpoint to a single protocol address, while permitting more than one connection to be accepted to the same protocol address. That means that although an attempt to bind a communications endpoint to some address with *qlen*=0 might be rejected with `TADDRBUSY`, the user may nevertheless use this (unbound) endpoint as a responding endpoint in a call to `t_accept(3NSL)`. To become independent of such implementation differences, the user should supply unbound responding endpoints to `t_accept(3NSL)`.

The local address bound to an endpoint may change as result of a `t_accept(3NSL)` or `t_connect(3NSL)` call. Such changes are not necessarily reversed when the connection is released.

t\_optmgmt(3NSL)

NAME	t_optmgmt – Manage options for a transport endpoint
SYNOPSIS	<pre>#include &lt;xti.h&gt;  int t_optmgmt(int fd, const struct t_optmgmt *req, struct t_optmgmt *ret);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the &lt;tiuser.h&gt; header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The t_optmgmt () function enables a transport user to retrieve, verify or negotiate protocol options with the transport provider. The argument <i>fd</i> identifies a transport endpoint.</p> <p>The <i>req</i> and <i>ret</i> arguments point to a t_optmgmt structure containing the following members:</p> <pre>struct netbuf opt; t_scalar_t      flags;</pre> <p>The <i>opt</i> field identifies protocol options and the <i>flags</i> field is used to specify the action to take with those options.</p> <p>The options are represented by a netbuf structure in a manner similar to the address in t_bind(3NSL). The argument <i>req</i> is used to request a specific action of the provider and to send options to the provider. The argument <i>len</i> specifies the number of bytes in the options, <i>buf</i> points to the options buffer, and <i>maxlen</i> 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>, <i>maxlen</i> specifies the maximum size of the options buffer and <i>buf</i> points to the buffer where the options are to be placed. If <i>maxlen</i> in <i>ret</i> is set to zero, no options values are returned. On return, <i>len</i> specifies the number of bytes of options returned. The value in <i>maxlen</i> 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.</p> <p>Each option in the options buffer is of the form struct t_opthdr possibly followed by an option value.</p> <p>The <i>level</i> field of struct t_opthdr identifies the XTI level or a protocol of the transport provider. The <i>name</i> field identifies the option within the level, and <i>len</i> contains its total length; that is, the length of the option header t_opthdr plus the length of the option value. If t_optmgmt () is called with the action T_NEGOTIATE set, the <i>status</i> field of the returned options contains information about the success or failure of a negotiation.</p> <p>Several options can be concatenated. The option user has, however to ensure that each options header and value part starts at a boundary appropriate for the architecture-specific alignment rules. The macros T_OPT_FIRSTHDR (nbp), T_OPT_NEXTHDR (nbp, tohp), T_OPT_DATA (tohp) are provided for that purpose.</p>

T\_OPT\_DATA (nhp)

If argument is a pointer to a t\_opthdr structure, this macro returns an unsigned character pointer to the data associated with the t\_opthdr.

T\_OPT\_NEXTHDR (nbp, tohp)

If the first argument is a pointer to a netbuf structure associated with an option buffer and second argument is a pointer to a t\_opthdr structure within that option buffer, this macro returns a pointer to the next t\_opthdr structure or a null pointer if this t\_opthdr is the last t\_opthdr in the option buffer.

T\_OPT\_FIRSTHDR (nbp)

If the argument is a pointer to a netbuf structure associated with an option buffer, this macro returns the pointer to the first t\_opthdr structure in the associated option buffer, or a null pointer if there is no option buffer associated with this netbuf or if it is not possible or the associated option buffer is too small to accommodate even the first aligned option header.

T\_OPT\_FIRSTHDR is useful for finding an appropriately aligned start of the option buffer. T\_OPT\_NEXTHDR is useful for moving to the start of the next appropriately aligned option in the option buffer. Note that T\_OPT\_NEXTHDR is also available for backward compatibility requirements. T\_OPT\_DATA is useful for finding the start of the data part in the option buffer where the contents of its values start on an appropriately aligned boundary.

If the transport user specifies several options on input, all options must address the same level.

If any option in the options buffer does not indicate the same level as the first option, or the level specified is unsupported, then the t\_optmgt () request will fail with TBADOPT. If the error is detected, some options have possibly been successfully negotiated. The transport user can check the current status by calling t\_optmgt () with the T\_CURRENT flag set.

The *flags* field of *req* must specify one of the following actions:

T\_NEGOTIATE

This action enables the transport user to negotiate option values.

The user specifies the options of interest and their values in the buffer specified by *req→opt.buf* and *req→opt.len*. The negotiated option values are returned in the buffer pointed to by *ret→opt.buf*. The *status* field of each returned option is set to indicate the result of the negotiation. The value is T\_SUCCESS if the proposed value was negotiated, T\_PARTSUCCESS if a degraded value was negotiated, T\_FAILURE if the negotiation failed (according to the negotiation rules), T\_NOTSUPPORT if the transport provider does not support this option or illegally requests negotiation of a privileged option, and T\_READONLY if modification of a read-only option was requested. If the status is T\_SUCCESS, T\_FAILURE, T\_NOTSUPPORT or T\_READONLY, the returned option value is the same as the one requested on input.

The overall result of the negotiation is returned in *ret→flags*.

## t\_optmgmt(3NSL)

This field contains the worst single result, whereby the rating is done according to the order T\_NOTSUPPORT, T\_READONLY, T\_FAILURE, T\_PARTSUCCESS, T\_SUCCESS. The value T\_NOTSUPPORT is the worst result and T\_SUCCESS is the best.

For each level, the option T\_ALLOPT can be requested on input. No value is given with this option; only the t\_opthdr part is specified. This input requests to negotiate all supported options of this level to their default values. The result is returned option by option in *ret→opt.buf*. Note that depending on the state of the transport endpoint, not all requests to negotiate the default value may be successful.

### T\_CHECK

This action enables the user to verify whether the options specified in *req* are supported by the transport provider. If an option is specified with no option value (it consists only of a t\_opthdr structure), the option is returned with its *status* field set to T\_SUCCESS if it is supported, T\_NOTSUPPORT if it is not or needs additional user privileges, and T\_READONLY if it is read-only (in the current XTI state). No option value is returned.

If an option is specified with an option value, the *status* field of the returned option has the same value, as if the user had tried to negotiate this value with T\_NEGOTIATE. If the status is T\_SUCCESS, T\_FAILURE, T\_NOTSUPPORT or T\_READONLY, the returned option value is the same as the one requested on input.

The overall result of the option checks is returned in *ret→flags*. This field contains the worst single result of the option checks, whereby the rating is the same as for T\_NEGOTIATE.

Note that no negotiation takes place. All currently effective option values remain unchanged.

### T\_DEFAULT

This action enables the transport user to retrieve the default option values. The user specifies the options of interest in *req→opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the t\_opthdr part of an option only. The default values are then returned in *ret→opt.buf*.

The *status* field returned is T\_NOTSUPPORT if the protocol level does not support this option or the transport user illegally requested a privileged option, T\_READONLY if the option is read-only, and set to T\_SUCCESS in all other cases. The overall result of the request is returned in *ret→flags*. This field contains the worst single result, whereby the rating is the same as for T\_NEGOTIATE.

For each level, the option T\_ALLOPT can be requested on input. All supported options of this level with their default values are then returned. In this case, *ret→opt.maxlen* must be given at least the value *info→options* before the call. See t\_getinfo(3NSL) and t\_open(3NSL).



**T\_CURRENT**

This action enables the transport user to retrieve the currently effective option values. The user specifies the options of interest in *req→opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the *t\_opthdr* part of an option only. The currently effective values are then returned in *req→opt.buf*.

The *status* field returned is **T\_NOTSUPPORT** if the protocol level does not support this option or the transport user illegally requested a privileged option, **T\_READONLY** if the option is read-only, and set to **T\_SUCCESS** in all other cases. The overall result of the request is returned in *ret→flags*. This field contains the worst single result, whereby the rating is the same as for **T\_NEGOTIATE**.

For each level, the option **T\_ALLOPT** can be requested on input. All supported options of this level with their currently effective values are then returned.

The option **T\_ALLOPT** can only be used with *t\_optmgt()* and the actions **T\_NEGOTIATE**, **T\_DEFAULT** and **T\_CURRENT**. It can be used with any supported level and addresses all supported options of this level. The option has no value; it consists of a *t\_opthdr* only. Since in a *t\_optmgt()* call only options of one level may be addressed, this option should not be requested together with other options. The function returns as soon as this option has been processed.

Options are independently processed in the order they appear in the input option buffer. If an option is multiply input, it depends on the implementation whether it is multiply output or whether it is returned only once.

Transport providers may not be able to provide an interface capable of supporting **T\_NEGOTIATE** and/or **T\_CHECK** functionalities. When this is the case, the error **TNOTSUPPORT** is returned.

The function *t\_optmgt()* may block under various circumstances and depending on the implementation. The function will block, for instance, if the protocol addressed by the call resides on a separate controller. It may also block due to flow control constraints; that is, if data sent previously across this transport endpoint has not yet been fully processed. If the function is interrupted by a signal, the option negotiations that have been done so far may remain valid. The behavior of the function is not changed if **O\_NONBLOCK** is set.

**RETURN VALUES**

*t\_optmgt()* returns:

- 0            On success.
- 1           On failure, and sets *t\_errno* to indicate the error.

**VALID STATES**

ALL - apart from **T\_UNINIT**.

**ERRORS**

On failure, *t\_errno* is set to one of the following:

- TBADF**            The specified file descriptor does not refer to a transport endpoint.
- TBADFLAG**        An invalid flag was specified.

t\_optmgmt(3NSL)

	TBADOPT	The specified options were in an incorrect format or contained illegal information.
	TBUFOVFLW	The number of bytes allowed for an incoming argument ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the value of that argument. The information to be returned in <i>ret</i> will be discarded.
	TNOTSUPPORT	This action is not supported by the transport provider.
	TOUTSTATE	The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.
	TPROTO	This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error ( <i>t_errno</i> ).
	TSYSERR	A system error has occurred during execution of this function, or the specified option requires <code>PRIV_NET_RAWACCESS</code> privilege.
<b>TLI COMPATIBILITY</b>	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.	
<b>Interface Header</b>	The XTI interfaces use the header file, <code>&lt;xti.h&gt;</code> . TLI interfaces should <i>not</i> use this header. They should use the header:  <code>#include &lt;tiuser.h&gt;</code>	
<b>Error Description Values</b>	The <i>t_errno</i> value TPROTO can be set by the XTI interface but not by the TLI interface.  The <i>t_errno</i> values that this routine can return under different circumstances than its XTI counterpart are TACCES and TBUFOVFLW.  TACCES                      can be returned to indicate that the user does not have permission to negotiate the specified options.  TBUFOVFLW                  can be returned even when the <i>maxlen</i> field of the corresponding buffer has been set to zero.	
<b>Option Buffers</b>	The format of the options in an <i>opt</i> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format. The macros <code>T_OPT_DATA</code> , <code>T_OPT_NEXTHDR</code> , and <code>T_OPT_FIRSTHDR</code> described for XTI are not available for use by TLI interfaces.	
<b>Actions</b>	The semantic meaning of various action values for the <i>flags</i> field of <i>req</i> differs between the TLI and XTI interfaces. TLI interface users should heed the following descriptions of the actions:  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	

t\_optmgmt(3NSL)

evaluate the requested options and negotiate the values, returning the negotiated values through *ret*.

T\_CHECK This action enables the user to verify whether the options specified in *req* are supported by the transport provider. On return, the *flags* field of *ret* 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 *ret*. In *req*, the *len* field of *opt* must be zero and the *buf* field may be NULL.

**Connectionless-Mode** 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.

For more information refer to the *Transport Interfaces Programming Guide*

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**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.

**Trusted Solaris 8  
4/01 Reference  
Manual**

t\_accept(3NSL), t\_bind(3NSL)

close(2), poll(2), select(3C), t\_alloc(3NSL), t\_close(3NSL),  
t\_connect(3NSL), t\_getinfo(3NSL), t\_listen(3NSL), t\_open(3NSL),  
t\_rcv(3NSL), t\_rcvconnect(3NSL), t\_rcvudata(3NSL), t\_snddis(3NSL),  
attributes(5)

*Transport Interfaces Programming Guide*

## t\_snd(3NSL)

<b>NAME</b>	t_snd – Send data or expedited data over a connection
<b>SYNOPSIS</b>	<pre>#include &lt;xti.h&gt;  int t_snd(int fd, void *buf, unsigned int nbytes, int flags);</pre>
<b>DESCRIPTION</b>	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the &lt;tiuser.h&gt; header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to send either normal or expedited data. The argument <i>fd</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 below:</p> <p><b>T_EXPEDITED</b>      If set in <i>flags</i>, the data will be sent as expedited data and will be subject to the interpretations of the transport provider.</p> <p><b>T_MORE</b>            If set in <i>flags</i>, this indicates 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 (or ETSDU).</p> <p>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(3NSL) or t_getinfo(3NSL), the T_MORE flag is not meaningful and will be ignored if set.</p> <p>The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU; that is, when the T_MORE flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs.</p> <p><b>T_PUSH</b>            If set in <i>flags</i>, requests that the provider transmit all data that it has accumulated but not sent. The request is a local action on the provider and does not affect any similarly named protocol flag (for example, the TCP PUSH flag). This effect of setting this flag is protocol-dependent, and it may be ignored entirely by transport providers which do not support the use of this feature.</p>

t\_snd(3NSL)

Note that the communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.

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_NONBLOCK` is set by means of `t_open(3NSL)` or `fcntl(2)`, `t_snd()` will execute in asynchronous mode, and will fail immediately if there are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared by means of either `t_look(3NSL)` or the EM interface.

On successful completion, `t_snd()` returns the number of bytes (octets) accepted by the communications provider. Normally this will equal the number of octets specified in `nbytes`. However, if `O_NONBLOCK` is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the communications provider. In this case, `t_snd()` returns a value that is less than the value of `nbytes`. If `t_snd()` is interrupted by a signal before it could transfer data to the communications provider, it returns `-1` with `t_errno` set to `TSYSERR` and `errno` set to `EINTR`.

If `nbytes` is zero and sending of zero bytes is not supported by the underlying communications service, `t_snd()` returns `-1` with `t_errno` set to `TBADDATA`.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as specified by the current values in the TSDU or ETSDU fields in the *info* argument returned by `t_getinfo(3NSL)`.

The error `TLOOK` is returned for asynchronous events. It is required only for an incoming disconnect event but may be returned for other events.

## RETURN VALUES

On successful completion, `t_snd()` returns the number of bytes accepted by the transport provider. Otherwise, `-1` is returned on failure and `t_errno` is set to indicate the error.

Note that if the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that `O_NONBLOCK` is set and the communications provider is blocked due to flow control, or that `O_NONBLOCK` is clear and the function was interrupted by a signal.

## ERRORS

On failure, `t_errno` is set to one of the following:

`TBADDATA`

Illegal amount of data:

- A single send was attempted specifying a TSDU (ETSDU) or fragment TSDU (ETSDU) greater than that specified by the current values of the TSDU or ETSDU fields in the *info* argument.
- A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider.

t\_snd(3NSL)

	<ul style="list-style-type: none"> <li>■ Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the <i>info</i> argument – the ability of an XTI implementation to detect such an error case is implementation-dependent. See WARNINGS, below.</li> </ul>
	<p>TBADF           The specified file descriptor does not refer to a transport endpoint.</p> <p>TBADFLAG       An invalid flag was specified.</p> <p>TFLOW           O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.</p> <p>TLOOK           An asynchronous event has occurred on this transport endpoint.</p> <p>TNOTSUPPORT     This function is not supported by the underlying transport provider.</p> <p>TOUTSTATE       The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</p> <p>TPROTO          This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>).</p> <p>TSYSERR         A system error has occurred during execution of this function.</p>
<b>TLI COMPATIBILITY</b>	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.
<b>Interface Header</b>	<p>The XTI interfaces use the header file, <i>&lt;xti.h&gt;</i>. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include &lt;tiuser.h&gt;</pre>
<b>Error Description Values</b>	<p>The <i>t_errno</i> values that can be set by the XTI interface and cannot be set by the TLI interface are:</p> <p>TPROTO TLOOK TBADFLAG TOUTSTATE</p> <p>The <i>t_errno</i> value that this routine can return under different circumstances than its XTI counterpart is TBADDATA.</p> <p>In the case of a TBADDATA error, TBADDATA is returned, only for illegal zero byte TSDU (ETSDU) send attempts.</p> <p>For more information refer to the <i>Transport Interfaces Programming Guide</i></p>

t\_snd(3NSL)

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY OF TRUSTED SOLARIS CHANGES** If the process calling these routines possesses the PRIV\_NET\_REPLY\_EQUAL privilege, the packets that 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.

Trusted Solaris 8  
4/01 Reference  
Manual  
SunOS 5.8  
Reference Manual

fcntl(2)

t\_getinfo(3NSL), t\_look(3NSL), t\_open(3NSL), t\_t\_rcv(3NSL),  
attributes(5)

*Transport Interfaces Programming Guide*

**WARNINGS** It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent t\_snd() calls then the different data may be intermixed.

Multiple sends which exceed the maximum TSDU or ETSDU size may not be discovered by XTI. In this case an implementation-dependent error will result, generated by the transport provider, perhaps on a subsequent XTI call. This error may take the form of a connection abort, a TSYSEERR, a TBADDDATA or a TPROTO error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, t\_snd() fails with TBADDDATA.

t\_sndudata(3NSL)

NAME	t_sndudata – Send a data unit
SYNOPSIS	<pre>#include &lt;xti.h&gt;  int t_sndudata(int fd, const struct t_unitdata *unitdata);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the &lt;tiuser.h&gt; header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used in connectionless-mode to send a data unit to another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be sent, and <i>unitdata</i> points to a <i>t_unitdata</i> structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; In <i>unitdata</i>, <i>addr</i> specifies the protocol address of the destination user, <i>opt</i> identifies options that the user wants associated with this request, and <i>udata</i> 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 <i>len</i> field of <i>opt</i> to zero. In this case, the provider uses the option values currently set for the communications endpoint.</pre> <p>If the <i>len</i> field of <i>udata</i> is zero, and sending of zero octets is not supported by the underlying transport service, the <i>t_sndudata()</i> will return set to TBADDATA.</p> <p>By default, <i>t_sndudata()</i> 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 <i>O_NONBLOCK</i> is set by means of <i>t_open(3NSL)</i> or <i>fcntl(2)</i>, <i>t_sndudata()</i> will execute in asynchronous mode and will fail under such conditions. The process can arrange to be notified of the clearance of a flow control restriction by means of either <i>t_look(3NSL)</i> or the EM interface.</p> <p>If the amount of data specified in <i>udata</i> exceeds the TSDU size as returned in the <i>tsdu</i> field of the <i>info</i> argument of <i>t_open(3NSL)</i> or <i>t_getinfo(3NSL)</i>, a TBADDATA error will be generated. If <i>t_sndudata()</i> is called before the destination user has activated its transport endpoint (see <i>t_bind(3NSL)</i>), the data unit may be discarded.</p> <p>If it is not possible for the transport provider to immediately detect the conditions that cause the errors TBADDADDR and TBADOPT, these errors will alternatively be returned by <i>t_rcvuderr</i>. Therefore, an application must be prepared to receive these errors in both of these ways.</p> <p>If the call is interrupted, <i>t_sndudata()</i> will return EINTR and the datagram will not be sent.</p>



<b>RETURN VALUES</b>	<p>t_sndudata() returns:</p> <p>0            On success.</p> <p>-1           On failure, and sets errno to indicate the error.</p>
<b>VALID STATES</b>	T_IDLE.
<b>ERRORS</b>	<p>On failure, t_errno is set to one of the following:</p> <p>TBADADDR        The specified protocol address was in an incorrect format or contained illegal information.</p> <p>TBADDATA        Illegal amount of data. A single send was attempted specifying a TSDU greater than that specified in the <i>info</i> argument, or a send of a zero byte TSDU is not supported by the provider.</p> <p>TBADF            The specified file descriptor does not refer to a transport endpoint.</p> <p>TBAADOPT        The specified options were in an incorrect format or contained illegal information.</p> <p>TFLOW            O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.</p> <p>TLOOK            An asynchronous event has occurred on this transport endpoint.</p> <p>TNOTSUPPORT     This function is not supported by the underlying transport provider.</p> <p>TOUTSTATE       The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</p> <p>TPROTO           This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).</p> <p>TSYSERR          A system error has occurred during execution of this function.</p>
<b>TLI COMPATIBILITY</b>	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.
<b>Interface Header</b>	<p>The XTI interfaces use the header file, &lt;xti.h&gt;. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include &lt;tiuser.h&gt;</pre>
<b>Error Description Values</b>	<p>The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:</p> <p>TPROTO</p> <p>TPROTO</p>

t\_sndudata(3NSL)

TBADADDR  
TBADOPT  
TLOOK  
TOUTSTATE

**Notes** Whenever this function fails with t\_error set to TFLOW, O\_NONBLOCK must have been set.

**Option Buffers** The format of the options in an opt buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY OF TRUSTED SOLARIS CHANGES** If the process calling these routines possesses the PRIV\_NET\_REPLY\_EQUAL privilege, the packets that 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.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**Trusted Solaris  
8/04 Reference  
Manual**

fcntl(2), t\_bind(3NSL)

t\_alloc(3NSL), t\_error(3NSL), t\_getinfo(3NSL), t\_look(3NSL),  
t\_open(3NSL), t\_rcvudata(3NSL), t\_rcvuderr(3NSL), attributes(5)

*Transport Interfaces Programming Guide*

tsol\_lbuild\_create(3TSOL)

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 [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/ModLabel.h&gt;  ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler)()     ok_callback lbuild_attributes extended_operation, .... , NULL);  void *tsol_lbuild_get(ModLabelData *data, lbuild_attributes     extended_operation);  void tsol_lbuild_set(ModLabelData *data lbuild_attributes     extended_operation, .... , NULL);  void tsol_lbuild_destroy(ModLabelData *data);</pre>						
DESCRIPTION	<p>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 <code>label_encodings(4)</code> 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.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, the label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>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 grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> 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 <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by the label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <table><tr><td><code>lbuild_dialog</code></td><td>The label builder dialog box.</td></tr><tr><td><code>ok</code></td><td>The OK pushbutton.</td></tr><tr><td><code>cancel</code></td><td>The Cancel pushbutton.</td></tr></table>	<code>lbuild_dialog</code>	The label builder dialog box.	<code>ok</code>	The OK pushbutton.	<code>cancel</code>	The Cancel pushbutton.
<code>lbuild_dialog</code>	The label builder dialog box.						
<code>ok</code>	The OK pushbutton.						
<code>cancel</code>	The Cancel pushbutton.						

tsol\_lbuild\_create(3TSOL)

reset                    The Reset pushbutton.

help                    The Help pushbutton.

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

widget                  The widget from which the dialog box is created. Any Motif widget can be passed.

ok\_callback            A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL              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 `ModLabelData` 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 information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

LBUILD\_MODE\_IL            Build an information label.

An information label is fixed at `ADMIN_LOW`.

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 buider 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.

tsol\_lbuild\_create(3TSOL)

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 supplied value 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 the label builder to create the user interface.

## EXAMPLES

### EXAMPLE 1 To create a Label Builder

```
(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);
```

### EXAMPLE 2 To query the mode and display the Label Builder

These examples call the tsol\_lbuild\_get () routine to query the mode being used, and call the tsol\_lbuild\_set () routine so the label builder dialog box displays.

**EXAMPLE 2** To query the mode and display the Label Builder (Continued)

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

**EXAMPLE 3** To destroy the ModLabelData variable

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  
Header file for label builder functions
  - /etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

- Trusted Solaris 8 4/01 Reference Manual** label\_encodings(4)  
Trusted Solaris Developer's Guide
- SunOS 5.8 Reference Manual** attributes(5)

tsol\_lbuild\_destroy(3TSOL)

<b>NAME</b>	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						
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/ModLabel.h&gt;  ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler)()     ok_callback lbuild_attributes extended_operation, .... , NULL);  void *tsol_lbuild_get(ModLabelData *data, lbuild_attributes     extended_operation);  void tsol_lbuild_set(ModLabelData *data lbuild_attributes     extended_operation, .... , NULL);  void tsol_lbuild_destroy(ModLabelData *data);</pre>						
<b>DESCRIPTION</b>	<p>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(4) 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.</p> <p>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 OK pushbutton widget.</p> <p>When choosing options, the 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.</p> <p>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 grayed. There are no privileges to override these restrictions.</p> <p>tsol_lbuild_create() creates the graphical user interface and returns a pointer variable of type ModLabeldata* 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 the 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.</p> <p>The widget information is accessed directly by referencing the following fields of the ModLabelData structure.</p> <table> <tr> <td>lbuild_dialog</td><td>The label builder dialog box.</td></tr> <tr> <td>ok</td><td>The OK pushbutton.</td></tr> <tr> <td>cancel</td><td>The Cancel pushbutton.</td></tr> </table>	lbuild_dialog	The label builder dialog box.	ok	The OK pushbutton.	cancel	The Cancel pushbutton.
lbuild_dialog	The label builder dialog box.						
ok	The OK pushbutton.						
cancel	The Cancel pushbutton.						



reset            The Reset pushbutton.

help            The Help pushbutton.

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

widget           The widget from which the dialog box is created. Any Motif widget can be passed.

ok\_callback       A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL        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 `ModLabelData` 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 information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

`LBUILD_MODE_IL`            Build an information label.

An information label is fixed at `ADMIN_LOW`.

`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`.

## tsol\_lbuild\_destroy(3TSOL)

### 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 buider 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.

tsol\_lbuild\_destroy(3TSOL)

**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 supplied value 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 the label builder to create the user interface.

**EXAMPLES**

**EXAMPLE 1** To create a Label Builder

```
(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);
```

**EXAMPLE 2** To query the mode and display the Label Builder

These examples call the tsol\_lbuild\_get() routine to query the mode being used, and call the tsol\_lbuild\_set() routine so the label builder dialog box displays.

tsol\_lbuild\_destroy(3TSOL)

**EXAMPLE 2** To query the mode and display the Label Builder (Continued)

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
NULL);
```

**EXAMPLE 3** To destroy the ModLabelData variable

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  
Header file for label builder functions

/etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

`label_encodings(4)`  
*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

`attributes(5)`

tsol\_lbuild\_get(3TSOL)

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 [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/ModLabel.h&gt;  ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler)()     ok_callback lbuild_attributes extended_operation, .... , NULL);  void *tsol_lbuild_get(ModLabelData *data, lbuild_attributes     extended_operation);  void tsol_lbuild_set(ModLabelData *data lbuild_attributes     extended_operation, .... , NULL);  void tsol_lbuild_destroy(ModLabelData *data);</pre>						
DESCRIPTION	<p>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(4) 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.</p> <p>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 OK pushbutton widget.</p> <p>When choosing options, the 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.</p> <p>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 grayed. There are no privileges to override these restrictions.</p> <p>tsol_lbuild_create() creates the graphical user interface and returns a pointer variable of type ModLabeldata* 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 the 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.</p> <p>The widget information is accessed directly by referencing the following fields of the ModLabelData structure.</p> <table><tr><td>lbuild_dialog</td><td>The label builder dialog box.</td></tr><tr><td>ok</td><td>The OK pushbutton.</td></tr><tr><td>cancel</td><td>The Cancel pushbutton.</td></tr></table>	lbuild_dialog	The label builder dialog box.	ok	The OK pushbutton.	cancel	The Cancel pushbutton.
lbuild_dialog	The label builder dialog box.						
ok	The OK pushbutton.						
cancel	The Cancel pushbutton.						

## tsol\_lbuild\_get(3TSOL)

reset                    The Reset pushbutton.

help                    The Help pushbutton.

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

widget                  The widget from which the dialog box is created. Any Motif widget can be passed.

ok\_callback             A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL               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 `ModLabelData` 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 information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

`LBUILD_MODE_IL`                    Build an information label.

An information label is fixed at `ADMIN_LOW`.

`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 buider 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.

tsol\_lbuild\_get(3TSOL)

**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 supplied value 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 the label builder to create the user interface.

**EXAMPLES**

**EXAMPLE 1** To create a Label Builder

```
(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);
```

**EXAMPLE 2** To query the mode and display the Label Builder

These examples call the tsol\_lbuild\_get() routine to query the mode being used, and call the tsol\_lbuild\_set() routine so the label builder dialog box displays.



**EXAMPLE 2** To query the mode and display the Label Builder (Continued)

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

**EXAMPLE 3** To destroy the ModLabelData variable

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  
Header file for label builder functions
  - /etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

- Trusted Solaris 8 4/01 Reference Manual** label\_encodings(4)  
Trusted Solaris Developer's Guide
- SunOS 5.8 Reference Manual** attributes(5)

tsol\_lbuild\_set(3TSOL)

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><b>cc</b> [<i>flag...</i>] <i>file...</i> -ltsol -lDtTsol [<i>library...</i>]  #include &lt;Dt/ModLabel.h&gt;  ModLabelData *<b>tsol_lbuild_create</b>(Widget <i>widget</i> void (*<i>event_handler</i>())     <i>ok_callback</i> <i>lbuild_attributes</i> <i>extended_operation</i>, .... , NULL) ;  void *<b>tsol_lbuild_get</b>(ModLabelData *<i>data</i>, <i>lbuild_attributes</i>     <i>extended_operation</i>) ;  void <b>tsol_lbuild_set</b>(ModLabelData *<i>data</i> <i>lbuild_attributes</i>     <i>extended_operation</i>, .... , NULL) ;  void <b>tsol_lbuild_destroy</b>(ModLabelData *<i>data</i>) ;</pre>						
DESCRIPTION	<p>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 <code>label_encodings(4)</code> 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.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, the label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>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 grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> 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 <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by the label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <table><tr><td><code>lbuild_dialog</code></td><td>The label builder dialog box.</td></tr><tr><td><code>ok</code></td><td>The OK pushbutton.</td></tr><tr><td><code>cancel</code></td><td>The Cancel pushbutton.</td></tr></table>	<code>lbuild_dialog</code>	The label builder dialog box.	<code>ok</code>	The OK pushbutton.	<code>cancel</code>	The Cancel pushbutton.
<code>lbuild_dialog</code>	The label builder dialog box.						
<code>ok</code>	The OK pushbutton.						
<code>cancel</code>	The Cancel pushbutton.						

reset            The Reset pushbutton.

help            The Help pushbutton.

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

widget           The widget from which the dialog box is created. Any Motif widget can be passed.

ok\_callback       A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL        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 `ModLabelData` 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 information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

`LBUILD_MODE_IL`            Build an information label.

An information label is fixed at `ADMIN_LOW`.

`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`.

## tsol\_lbuild\_set(3TSOL)

### 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 buider 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 supplied value 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 the label builder to create the user interface.

**EXAMPLES****EXAMPLE 1** To create a Label Builder

```
(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);
```

**EXAMPLE 2** To query the mode and display the Label Builder

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

tsol\_lbuild\_set(3TSOL)

**EXAMPLE 2** To query the mode and display the Label Builder (Continued)

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

**EXAMPLE 3** To destroy the ModLabelData variable

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  
Header file for label builder functions

/etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

`label_encodings(4)`  
*Trusted Solaris Developer's Guide*

**SunOS 5.8  
Reference Manual**

`attributes(5)`

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database 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</p>

## updwtmp(3C)

	<p><i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i>⇒<code>ut_id</code>. If the end of database is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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>⇒<code>ut_line</code> string. If the end of the database is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open database.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)</p>



getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
<b>RETURN VALUES</b>	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

## updwtmp(3C)

	The <code>endutxent()</code> and <code>setutxent()</code> functions return no value.				
	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.				
USAGE	These functions use buffered standard I/O for input, but <code>pututxline()</code> uses an unbuffered write to avoid race conditions between processes trying to modify the <code>utmpx</code> and <code>wtmpx</code> files.				
	Applications should not access the <code>utmpx</code> and <code>wtmpx</code> databases directly, but should use these functions to ensure that these databases are maintained consistently.				
FILES	<table><tr><td><code>/var/adm/utmpx</code></td><td>User access and accounting information</td></tr><tr><td><code>/var/adm/wtmpx</code></td><td>History of user access and accounting information</td></tr></table>	<code>/var/adm/utmpx</code>	User access and accounting information	<code>/var/adm/wtmpx</code>	History of user access and accounting information
<code>/var/adm/utmpx</code>	User access and accounting information				
<code>/var/adm/wtmpx</code>	History of user access and accounting information				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES	<code>pututxline()</code> invokes a program with appropriate forced privileges to verify and write the <code>utmpx</code> structure. <code>pututxline</code> clears fields in an entry if the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute
Trusted Solaris 8 4/01 Reference Manual	<code>getutent(3C)</code>
Notes Reference Manual	<code>wait(2)</code> , <code>ttyslot(3C)</code> , <code>utmpx(4)</code> , <code>attributes(5)</code> , <code>wstat(3XFN)</code>
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 <code>getutxid()</code> or <code>getutxline()</code> , 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 <code>getutxline()</code> to search for multiple occurrences it would be necessary to zero out the static after each success, or <code>getutxline()</code> 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 <code>pututxline()</code> (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 <code>getutxent()</code> , <code>getutxid()</code> , or <code>getutxline()</code> routines, if the user has just modified those contents and passed the pointer back to <code>pututxline()</code> .

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database 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</p>

## updwtmpx(3C)

	<p><i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i>⇒<code>ut_id</code>. If the end of database is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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>⇒<code>ut_line</code> string. If the end of the database is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open database.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)</p>

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions).</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

## updwtmpx(3C)

The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference** `wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

**Notes**  
**Reference Manual**  
**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()`.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include &lt;utmp.h&gt;  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>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char          ut_user[8];      /* user login name */ char          ut_id[4];       /* /sbin/inittab id (usually line #) */ char          ut_line[12];    /* device name (console, lnxx) */ 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> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short  e_termination;    /* termination status */ short  e_exit;           /* exit status */</pre> <p><code>getutent()</code> The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p><code>getutid()</code> The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id⇒ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id⇒ut_id</code>. If the end of file is reached without a match, it fails.</p> <p><code>getutline()</code> The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line⇒ut_line</code> string. If the end of file is reached without a match, it fails.</p> <p><code>pututline()</code> The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already</p>

## utmpname(3C)

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 functions. 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` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member 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()` The `setutent()` function 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()` The `endutent()` function closes the currently open file.

`utmpname()` The `utmpname()` function 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. The `utmpname()` function does not open the file but 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.

**USAGE** These functions 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` files.

Applications should not access the `utmp` and `wtmp` databases directly, but should use these functions to ensure that these databases are maintained consistently. Using these functions, however, may cause applications to fail if user accounting data cannot be represented properly in the `utmp` structure (for example, on a system where PIDs can exceed 32767). Use the functions described on the `getutxent(3C)` manual page instead.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe



utmpname(3C)

**SUMMARY OF  
TRUSTED  
SOLARIS  
CHANGES**  
**SunOS 5.8  
Reference Manual**  
**NOTES**

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.

ttyslot(3C), utmp(4), utmpx(4), attributes(5)

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 function 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() functions, if the user has just modified those contents and passed the pointer back to pututline().

utmpxname(3C)

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – User accounting database functions
SYNOPSIS	<pre>#include &lt;utmpx.h&gt;  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>
DESCRIPTION	<p>These functions provide access to the user accounting database, utmpx (see utmpx(4)). Entries in the database are described by the definitions and data structures in &lt;utmpx.h&gt;.</p> <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> <p>The structure exit status includes the following members:</p> <pre>short  e_termination; /* termination status */ short  e_exit;         /* exit status */</pre> <p>getutxent()</p> <p>The getutxent() function reads in the next entry from a utmpx database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.</p> <p>getutxid()</p> <p>The getutxid() function searches forward from the current point in the utmpx database until it finds an entry with a ut_type matching id⇒ut_type, if the type specified is RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME. If the type specified in</p>

	<i>id</i> is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of database is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> database 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> ⇒ <code>ut_line</code> string. If the end of the database is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> 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 <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of ADMIN_LOW, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of ADMIN_LOW. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either USER_PROCESS or DEAD_PROCESS; and the <code>ut_line</code> member 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.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open database.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the database file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but 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 <code>utmp</code> file to be easily obtainable.; otherwise, an error value of 0 is returned. The function returns 1 on success.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is silently truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)

## utmpxname(3C)

getutmpx()	The getutmpx() function copies the information stored in the members of the utmp structure to the corresponding members of the utmpx structure. (See getutent(3C) for utmp structure)
updwtmp()	<p>The updwtmp() function can be used in two ways.</p> <p>If <i>wfile</i> is /var/adm/wtmp, the utmp format record supplied by the caller is converted to a utmpx format record and the /var/adm/wtmpx file is updated (because the /var/adm/wtmp file no longer exists, operations on wtmp are converted to operations on wtmpx by the library functions.</p> <p>If <i>wfile</i> is a file other than /var/adm/wtmp, it is assumed to be an old file in utmp format and is updated directly with the utmp format record supplied by the caller.</p>
updwtmpx()	The updwtmpx() function writes the contents of the utmpx structure pointed to by <i>utmpx</i> to the database.
utmpx structure	<p>The values of the e_termination and e_exit members of the ut_exit structure are valid only for records of type DEAD_PROCESS. For utmpx entries created by init(1M), these values are set according to the result of the wait() call that init performs on the process when the process exits. See the wait(2) manual page for the values init uses. Applications creating utmpx entries can set ut_exit values using the following code example:</p> <pre>u-&gt;ut_exit.e_termination = WTERMSIG(process-&gt;p_exit) u-&gt;ut_exit.e_exit = WEXITSTATUS(process-&gt;p_exit)</pre> <p>See wstat(3XFN) for descriptions of the WTERMSIG and WEXITSTATUS macros.</p> <p>The ut_session member is not acted upon by the operating system. It is used by applications interested in creating utmpx entries.</p> <p>For records of type USER_PROCESS, the nonuser() and nonuserx() macros use the value of the ut_exit.e_exit member to mark utmpx entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each pty to have a utmpx record (as most applications expect.). The NONROOT_USER macro defines the value that login places in the ut_exit.e_exit member.</p>
RETURN VALUES	<p>Upon successful completion, getutxent(), getutxid(), and getutxline() each return a pointer to a utmpx structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.</p> <p>The return value may point to a static area which is overwritten by a subsequent call to getutxid() or getutxline().</p> <p>Upon successful completion, pututxline() returns a pointer to a utmpx structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.</p>

The `endutxent()` and `setutxent()` functions return no value.

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

**USAGE** These functions 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.

Applications should not access the `utmpx` and `wtmpx` databases directly, but should use these functions to ensure that these databases are maintained consistently.

**FILES**

<code>/var/adm/utmpx</code>	User access and accounting information
<code>/var/adm/wtmpx</code>	History of user access and accounting information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**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

**Trusted Solaris 8** `getutent(3C)`

**4/01 Reference**

**Manual**

**Reference Manual**

**NOTES**

`wait(2)`, `ttyslot(3C)`, `utmpx(4)`, `attributes(5)`, `wstat(3XFN)`

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()`.

## Xbcleartos(3TSOL)

<b>NAME</b>	labelclipping, Xbcltos, Xbsltos, Xbcleartos – translate a binary label and clip to the specified width
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/label_clipping.h&gt;  XmString <b>Xbcltos</b>(Display *display, const bclabel_t *cmwlabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbsltos</b>(Display *display, const bxlabel_t *senslabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbcleartos</b>(Display *display, const bclear_t *clearance,     Dimension width, const XmFontList fontlist, const int flags);</pre>
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p><i>display</i>                      The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i>                    The CMW label to be translated.</p> <p><i>senslabel</i>                   The sensitivity label to be translated.</p> <p><i>clearance</i>                  The clearance to be translated.</p> <p><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&lt;-. 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.</p> <p><i>fontlist</i>                    A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i>                      The value of flags indicates which words in the label_encodings(4) 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].</p>
<b>RETURN VALUES</b>	These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity 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(4)* file, or not dominated by the process' sensitivity label and the PRIV\_SYS\_TRANS\_LABEL privilege is not asserted.

**FILES** /usr/dt/include/Dt/label\_clipping.h  
Header file for label clipping functions

/etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**EXAMPLES** **EXAMPLE 1** To Translate and Clip a Clearance

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

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

**ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual** bltos(3TSOL), label\_encodings(4)

*Trusted Solaris Developer's Guide*

*Trusted Solaris Label Administration*

**SunOS 5.8  
Reference Manual** attributes(5)

See XmStringDraw(3) and FontList(3) for information on the creation and structure of a font list.

## Xbcltos(3TSOL)

<b>NAME</b>	labelclipping, Xbcltos, Xbsltos, Xbcleartos – translate a binary label and clip to the specified width
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/label_clipping.h&gt;  XmString <b>Xbcltos</b>(Display *display, const bclabel_t *cmwlabel,                 Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbsltos</b>(Display *display, const bxlabel_t *senslabel,                 Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbcleartos</b>(Display *display, const bclear_t *clearance,                 Dimension width, const XmFontList fontlist, const int flags);</pre>
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p><i>display</i>                      The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i>                    The CMW label to be translated.</p> <p><i>senslabel</i>                   The sensitivity label to be translated.</p> <p><i>clearance</i>                   The clearance to be translated.</p> <p><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&lt;-. 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.</p> <p><i>fontlist</i>                    A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i>                        The value of flags indicates which words in the label_encodings(4) 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].</p>
<b>RETURN VALUES</b>	These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity 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(4)* file, or not dominated by the process' sensitivity label and the PRIV\_SYS\_TRANS\_LABEL privilege is not asserted.

**FILES**

*/usr/dt/include/Dt/label\_clipping.h*  
Header file for label clipping functions

*/etc/security/tsol/label\_encodings*  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**EXAMPLES****EXAMPLE 1** To Translate and Clip a Clearance

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

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

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

*bltos(3TSOL)*, *label\_encodings(4)*

*Trusted Solaris Developer's Guide*

*Trusted Solaris Label Administration*

**SunOS 5.8  
Reference Manual**

*attributes(5)*

See *XmStringDraw(3)* and *FontList(3)* for information on the creation and structure of a font list.

## Xbsltos(3TSOL)

<b>NAME</b>	labelclipping, Xbcltos, Xbsltos, Xbcleartos – translate a binary label and clip to the specified width
<b>SYNOPSIS</b>	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]  #include &lt;Dt/label_clipping.h&gt;  XmString <b>Xbcltos</b>(Display *display, const bclabel_t *cmwlabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbsltos</b>(Display *display, const bxlabel_t *senslabel,     Dimension width, const XmFontList fontlist, const int flags);  XmString <b>Xbcleartos</b>(Display *display, const bclear_t *clearance,     Dimension width, const XmFontList fontlist, const int flags);</pre>
<b>DESCRIPTION</b>	<p>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' sensitivity label.</p> <p><i>display</i>                      The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i>                    The CMW label to be translated.</p> <p><i>senslabel</i>                   The sensitivity label to be translated.</p> <p><i>clearance</i>                  The clearance to be translated.</p> <p><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&lt;-. 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.</p> <p><i>fontlist</i>                    A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i>                      The value of flags indicates which words in the label_encodings(4) 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].</p>
<b>RETURN VALUES</b>	These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity 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(4)* file, or not dominated by the process' sensitivity label and the PRIV\_SYS\_TRANS\_LABEL privilege is not asserted.

**FILES**

/usr/dt/include/Dt/label\_clipping.h  
Header file for label clipping functions

/etc/security/tsol/label\_encodings  
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.

**EXAMPLES****EXAMPLE 1** To Translate and Clip a Clearance

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

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

**ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

**Trusted Solaris 8  
4/01 Reference  
Manual**

*bltos(3TSOL)*, *label\_encodings(4)*

*Trusted Solaris Developer's Guide*

*Trusted Solaris Label Administration*

**SunOS 5.8  
Reference Manual**

*attributes(5)*

See *XmStringDraw(3)* and *FontList(3)* for information on the creation and structure of a font list.

## xprt\_register(3NSL)

<b>NAME</b>	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister – Library routines for registering servers
<b>DESCRIPTION</b>	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
<b>Routines</b>	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xprt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i>⇒<i>nc_netid</i>] to <i>xprt</i>⇒<i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

xprt\_register(3NSL)

If the server has the PRIV\_NET\_MAC\_READ privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the PRIV\_NET\_PRIVADDR privilege.

The PRIV\_NET\_SETID privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

int svc\_auth\_reg(const int cred\_flavor, const enum auth\_stat (\*handler)());

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

void xprt\_register(const SVCXPRT \*xprt);

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

void xprt\_unregister(const SVCXPRT \*xprt);

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

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

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, `svc_unreg()` must be called with the privilege on.

xprt\_register(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

**Trusted Solaris 8  
4/01 Reference  
Manual**  
**SunOS 5.8  
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`

xprt\_unregister(3NSL)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister – Library routines for registering servers
DESCRIPTION	These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code> ], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.
Routines	<p>See <code>rpc(3NSL)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include &lt;rpc/rpc.h&gt;  bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (*procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char *nettype);</pre> <p>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 <code>static</code> 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 while <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 <code>rpc(3NSL)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>int svc_reg(const SVCXPRT *xprt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf);</pre> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i>⇒<i>nc_netid</i>] to <i>xprt</i>⇒<i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> 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 <code>PRIV_NET_PRIVADDR</code> privilege.</p> <pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum);</pre> <p>Remove from the <code>rpcbind</code> 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 [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p>

## xprt\_unregister(3NSL)

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

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

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

Registers the service authentication routine *handler* with the dispatch mechanism so that it 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`, `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, and -1 otherwise.

```
void xprt_register(const SVCXPRT *xprt);
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3NSL)`). Service implementors usually do not need this routine.

```
void xprt_unregister(const SVCXPRT *xprt);
```

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

### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### 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, `svc_unreg()` must be called with the privilege on.



xprt\_unregister(3NSL)

The PRIV\_NET\_PRIVADDR privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The PRIV\_NET\_SETID privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

`inetd(1M)`, `rpcbind(1M)`, `rpc(3NSL)`, `rpc_svc_calls(3NSL)`,  
`rpc_svc_create(3NSL)`, `rpcbind(3NSL)`

`select(3C)`, `rpc_svc_err(3NSL)`, `attributes(5)`

## XTSOLgetClientAttributes(3)

NAME	XTSOLgetClientAttributes – Get all CMW attributes associated with a client							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLgetClientAttributes</b> (<i>display</i>, <i>windowid</i>, <i>clientattrp</i>) ;  Display *<i>display</i>; XID <i>windowid</i>; XTSOLClientAttributes *<i>clientattrp</i>;</pre>							
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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid client						
SEE ALSO	XTSOLgetPropAttributes(3), XTSOLgetResAttributes(3)							

## XTSOLgetPropAttributes(3)

NAME	XTSOLgetPropAttributes – Get all CMW attributes associated with a property hanging on a window							
SYNOPSIS	#include <tsol/Xtsol.h>  Status <b>XTSOLgetPropAttributes</b> ( <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> ;							
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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay ().						
	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetClientAttributes(3), XTSOLgetResAttributes(3)							

## XTSOLgetPropLabel(3)

NAME	XTSOLgetPropLabel – Get the CMW label associated with a property hanging on a window							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLgetPropLabel</b> (<i>display</i>, <i>window</i>, <i>property</i>, <i>cmwlabel</i>) ;  Display *<i>display</i>; Window <i>window</i>; Atom <i>property</i>; bclabel_t *<i>cmwlabel</i>;</pre>							
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.							
PARAMETERS	<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 an SL. 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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetPropAttributes(3), XTSOLsetPropLabel(3)							

NAME	XTSOLgetPropUID – Get the UID associated with a property hanging on a window								
SYNOPSIS	#include <tsol/Xtsol.h>  Status <b>XTSOLgetPropUID</b> ( <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> ;								
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.								
PARAMETERS	<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.							
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>			ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWxwplt (available only on Trusted Solaris systems)								
MT-Level	MT-Unsafe								
RETURN VALUES	None								
ERRORS	BadAccess	Lack of privilege							
	BadWindow	Not a valid window							
	BadAtom	Not a valid atom							
SEE ALSO	XTSOLgetPropAttributes(3), XTSOLsetPropUID(3)								

## XTSOLgetResAttributes(3)

NAME	XTSOLgetResAttributes – Get all CMW attributes associated with a window or a pixmap							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLgetResAttributes</b> (<i>display, object, type, cmwwinattrp</i>) ;  Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; XTSOLResAttributes *<i>cmwwinattrp</i>;</pre>							
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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
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	XTSOLgetClientAttributes(3), XTSOLgetPropAttributes(3)							

NAME	XTSOLgetResLabel – Get the CMW label associated with a window, a pixmap, or a colormap							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLgetResLabel</b> (<i>display</i>, <i>object</i>, <i>type</i>, <i>cmwlabel</i>) ;  Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; bclabel_t *<i>cmwlabel</i>;</pre>							
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.							
PARAMETERS	<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 an SL. 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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid type						
SEE ALSO	XTSOLgetClientAttributes(3), XTSOLsetResLabel(3)							

## XTSOLgetResUID(3)

NAME	XTSOLgetResUID – Get the UID associated with a window, a pixmap							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLgetResUID</b>(<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>							
DESCRIPTION	<p>The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges.</p> <p>XTSOLgetResUID() gets the ownership of a window system object. This allows a client to get the ownership of an object it did not create.</p>							
PARAMETERS	<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.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></tbody></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
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	XTSOLgetClientAttributes(3), XTSOLgetResAttributes(3), XTSOLgetResUID(3)							



NAME	XTSOLgetSSHeight – Get the height of screen stripe							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLgetSSHeight</b> (<i>display</i>, <i>screen_num</i>, <i>newheight</i>) ;  Display *<i>display</i>; int <i>screen_num</i>; int *<i>newheight</i>;</pre>							
DESCRIPTION	XTSOLgetSSHeight () 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.							
PARAMETERS	<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 storage area where the height of the stripe in pixels is returned.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i>						
SEE ALSO	XTSOLsetSSHeight(3)							

## XTSOLgetWorkstationOwner(3)

NAME	XTSOLgetWorkstationOwner – Get the ownership of the workstation							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLgetWorkstationOwner</b> (<i>display</i>, <i>uidp</i>) ;  Display *<i>display</i>; uid_t *<i>uidp</i>;</pre>							
DESCRIPTION	XTSOLgetWorkstationOwner() is used to get the ownership of the workstation.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None.							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLsetWorkstationOwner(3)							

NAME	XTSOLIsWindowTrusted – Test if a window is created by a trusted client							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLIsWindowTrusted</b>(<i>display</i>, <i>window</i>) ;  Display *<i>display</i>; Window *<i>window</i>;</pre>							
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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	True	if the window is created by a trusted client.						
ERRORS	BadWindow	Not a valid window						

## XTSOLMakeTPWindow(3)

NAME	XTSOLMakeTPWindow – Make this window a Trusted Path window							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLMakeTPWindow</b>(<i>display</i>, <i>w</i>) ;  Display *<i>display</i>; Window *<i>w</i>;</pre>							
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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>w</i>	Specifies the ID of a window.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadValue	Not a valid type						

NAME	XTSOLsetPolyInstInfo – Set polyinstantiation information							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLsetPolyInstInfo</b>(<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>							
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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid <i>display</i> or <i>sl</i> .						

## XTSOLsetPropLabel(3)

NAME	XTSOLsetPropLabel – Set the CMW label associated with a property hanging on a window							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLsetPropLabel</b> (<i>*display, window, property, *cmwlabel, 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>							
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, and PRIV_WIN_UPGRADE_SL privileges.							
PARAMETERS	<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 and SETCL_SL.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
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	XTSOLgetPropAttributes(3), XTSOLgetPropLabel(3)							

NAME	XTSOLsetPropUID – Set the UID associated with a property hanging on a window							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLsetPropUID</b>(<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>							
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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetPropAttributes(3), XTSOLgetPropUID(3)							

## XTSOLsetResLabel(3)

NAME	XTSOLsetResLabel – Set the CMW label associated with a window or a pixmap							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLsetResLabel</b> (<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 <b>setting_flag</b> (<i>labelFlag</i>) ;</pre>							
DESCRIPTION	<p>The client must have the PRIV_WIN_DAC_WRITE, PRIV_WIN_MAC_WRITE, PRIV_WIN_UPGRADE_SL, and PRIV_WIN_DOWNGRADE_SL privileges.</p> <p>XTSOLsetResLabel () is used to change the CMW label associated with a window or a pixmap.</p>							
PARAMETERS	<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, and RES_SL.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
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	XTSOLgetResAttributes(3), XTSOLgetResLabel(3)							



NAME	XTSOLsetResUID – Set the UID associated with a window, a pixmap, or a colormap							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  Status <b>XTSOLsetResUID</b>(<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>							
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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
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(3)							

## XTSOLsetSessionHI(3)

NAME	XTSOLsetSessionHI – Set the session high sensitivity label to the window server								
SYNOPSIS	#include <tsol/Xtsol.h>  XTSOLsetSessionHI (display, sl) ;  Display *display; bslabel_t *sl;								
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.								
PARAMETERS	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 HIGH label.							
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:								
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>			ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWxwplt (available only on Trusted Solaris systems)								
MT-Level	MT-Unsafe								
RETURN VALUES	None								
ERRORS	BadAccess	Lack of privilege							
SEE ALSO	XTSOLsetSessionLO(3)								

NAME	XTSOLsetSessionLO – Set the session low sensitivity label to the window server							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLsetSessionLO</b>(<i>display</i>, <i>sl</i>) ;  Display *<i>display</i>; bslabel_t *<i>sl</i>;</pre>							
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.							
PARAMETERS	<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 low label.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLsetSessionHI(3)							

## XTSOLsetSSHeight(3)

NAME	XTSOLsetSSHeight – Set the height of screen stripe							
SYNOPSIS	<pre>#include &lt;tsol/Xtsol.h&gt;  XTSOLsetSSHeight (display, screen_num, newheight) ;  Display *display; int screen_num; int newheight;</pre>							
DESCRIPTION	XTSOLsetSSHeight ( ) 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.							
PARAMETERS	<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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i> .						
SEE ALSO	XTSOLgetSSHeight(3)							

NAME	XTSOLsetWorkstationOwner – Set the ownership of the workstation							
SYNOPSIS	#include <tsol/Xtsol.h>  <b>XTSOLsetWorkstationOwner</b> ( <i>display</i> , <i>uidp</i> ) ;  Display * <i>display</i> ; uid_t * <i>uidp</i> ; XTSOLClientAttributes * <i>clientattrp</i> ;							
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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>uidp</i>	Specifies a pointer to a uid_t structure that contains a UID.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLgetWorkstationOwner(3)							

## XTSOLShutdown(3)

<b>NAME</b>	XTSOLShutdown – Shut down the system
<b>SYNOPSIS</b>	<pre>#include &lt;tsol/Xtsol.h&gt;  <b>XTSOLShutdown</b>(<i>display</i>) ;  Display * <i>display</i></pre>
<b>DESCRIPTION</b>	XTSOLShutdown() shuts down the system of <i>display</i> . The client must have the PRIV_SYS_BOOT privilege to perform this operation.
<b>PARAMETERS</b>	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay().
<b>ATTRIBUTES</b>	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxwplt (available only on Trusted Solaris systems)
MT-Level	MT-Unsafe

# Index

---

## A

`accept` — accept a connection on a socket, 38  
`adornfc` — adorn the final component of a  
    pathname, 40  
`audit` control file information  
    — `endac`, 282, 338, 340, 342, 344, 346, 865  
    — `getacdir`, 282, 338, 340, 342, 344, 346, 865  
    — `getacflg`, 282, 338, 340, 342, 344, 346, 865  
    — `getacinfo`, 282, 338, 340, 342, 344, 346, 865  
    — `getacmin`, 282, 338, 340, 342, 344, 346, 865  
    — `getacna`, 282, 338, 340, 342, 344, 346, 865  
    — `setac`, 282, 338, 340, 342, 344, 346, 865  
`audit` record tokens, manipulating  
    — `au_preselect`, 54  
`auditwrite` — construct user-level audit  
    records, 42  
`au_preselect` — preselect an audit record, 54  
`au_user_mask` — get user's binary preselection  
    mask, 58  
`aw_errno` — obtain and display auditwrite error  
    messages, 60, 62, 64, 66, 68  
`aw_geterrno` — obtain and display auditwrite  
    error messages, 60, 62, 64, 66, 68  
`aw_perror` — obtain and display auditwrite  
    error messages, 60, 62, 64, 66, 68  
`aw_perror_r` — obtain and display auditwrite  
    error messages, 60, 62, 64, 66, 68  
`aw_strerror` — obtain and display auditwrite  
    error messages, 60, 62, 64, 66, 68

## B

`bclearhigh` — initialize admin high binary  
    clearance, 70, 72, 81, 85, 87, 100, 110, 127,  
    129, 138  
`bclearlow` — initialize admin low binary  
    clearance, 70, 72, 81, 85, 87, 100, 110, 127,  
    129, 138  
`bcleartoh` — binary clearance to hexadecimal  
    string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`bcleartoh_r` — binary clearance to hexadecimal  
    string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`bcleartos` — binary clearance to string, 78, 96,  
    121, 135  
`bclearundef` — initialize undefined binary  
    clearance, 70, 72, 81, 85, 87, 100, 110, 127,  
    129, 138  
`bclearvalid` — check validity of binary  
    clearance, 83, 125, 140  
`bclhigh` — initialize admin high binary CMW  
    label, 70, 72, 81, 85, 87, 100, 110, 127, 129,  
    138  
`bcllow` — initialize admin low binary CMW  
    label, 70, 72, 81, 85, 87, 100, 110, 127, 129,  
    138  
`bcltobanner` — translate binary CMW label to  
    printer banner page fields, 89  
`bcltoh` — binary CMW label to hexadecimal  
    string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`bcltoh_r` — binary CMW label to hexadecimal  
    string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`bcltos` — binary CMW label to string, 78, 96,  
    121, 135

**bcltosl** — reference sensitivity label, 99, 115, 391, 878  
**bclundef** — initialize undefined binary CMW label, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
**bind** — bind a name to a socket, 102  
**bind an address to a transport endpoint** — **t\_bind**, 1074  
**blcompare** — compare labels, 104, 105, 106, 107, 116  
**bldominates** — compare levels for dominance, 104, 105, 106, 107, 116  
**blequal** — compare levels for equality, 104, 105, 106, 107, 116  
**blinrange** — compare level to be between bounding levels, 104, 105, 106, 107, 116  
**blinset** — check level for set inclusion, 108  
**blmanifest** — create manifest binary labels, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
**blmaximum** — least upper bound of two binary levels, 112, 113, 114  
**blminimum** — greatest lower bound two binary levels, 112, 113, 114  
**blportion** — access binary labels, 99, 115, 391, 878  
**blstrictdom** — compare levels for strict dominance, 104, 105, 106, 107, 116  
**bltcolor** — get character-coded color name of label, 117, 119  
**bltcolor\_r** — get character-coded color name of label, 117, 119  
**bltype** — check type of binary label, 124, 877  
**blvalid** — check validity of binary label, 83, 125, 140  
**bslhigh** — initialize admin high binary sensitivity label, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
**bsllow** — initialize admin low binary sensitivity label, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
**bsltoh** — binary sensitivity label to hexadecimal string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
**bsltoh\_r** — binary sensitivity label to hexadecimal string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
**bsltos** — binary sensitivity label to string, 78, 96, 121, 135

**bslundef** — initialize undefined binary sensitivity label, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
**bslvalid** — check validity of binary sensitivity label, 83, 125, 140  
**btohex** — convert binary label to hexadecimal, 74, 76, 92, 94, 131, 133, 142, 463, 472

## C

**chkauthattr** — verify user authorization, 145, 289, 318, 380, 383, 872  
**clnt\_call** — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
**clnt\_control** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_create** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_create\_timed** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_create\_vers** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_create\_vers\_timed** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_destroy** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
**clnt\_dg\_create** — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830



`clnt_freeres` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_geterr` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_pcreateerror` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_perrno` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_perror` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_raw_create` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_spccreateerror` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_sperrno` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_sperror` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
`clnt_tli_create` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_tp_create` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_tp_create_timed` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clnt_vc_create` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
`clock_getres` — high-resolution clock operations, 260, 262, 264  
`clock_gettime` — high-resolution clock operations, 260, 262, 264  
`clock_settime` — high-resolution clock operations, 260, 262, 264  
communications  
    accept a connection on a socket — `accept`, 38  
    bind a name to a socket — `bind`, 102  
    create an endpoint for communication — `socket`, 906  
    listen for connections on a socket — `listen`, 502  
    send a message from a socket — `send`, `sendto`, `sendmsg`, 859, 861, 863  
construct user-level audit records — `auditwrite`, 42  
convert binary clearance to canonical string — `sbclear`, 851, 853, 855, 857  
convert binary CMW label to canonical string — `sbcl`, 851, 853, 855, 857  
convert binary sensitivity label to canonical string — `sbsl`, 851, 853, 855, 857  
convert hexadecimal string to binary clearance — `hextobclear`, 471, 480, 481, 482  
convert hexadecimal string to binary CMW label — `hextobcl`, 471, 480, 481, 482  
convert hexadecimal string to binary label — `hextob`, 471, 480, 481, 482  
convert hexadecimal string to binary sensitivity label — `hextobsl`, 471, 480, 481, 482  
convert string to binary clearance — `stobclear`, 909, 912, 915, 918  
convert string to binary CMW label — `stobcl`, 909, 912, 915, 918  
convert string to binary sensitivity label — `stobsl`, 909, 912, 915, 918  
create a door descriptor — `door_create`, 278  
current working directory  
    get pathname — `mldmldgetcwd`, 507

## D

### directories

get pathname of current working directory  
— mldgetcwd, 507

### display auditwrite error messages

— aw\_errno, 60, 62, 64, 66, 68  
— aw\_geterrno, 60, 62, 64, 66, 68  
— aw\_perror, 60, 62, 64, 66, 68  
— aw\_perror\_r, 60, 62, 64, 66, 68  
— aw\_strerror, 60, 62, 64, 66, 68

dn\_comp — resolver routines, 266, 272, 312,  
465, 474, 679, 685, 691, 697, 703, 709, 715, 721,  
727, 733, 739, 745, 751, 757, 763, 769

dn\_expand — resolver routines, 266, 272, 312,  
465, 474, 679, 685, 691, 697, 703, 709, 715, 721,  
727, 733, 739, 745, 751, 757, 763, 769

door\_create — create a door descriptor, 278

door\_tcred — return extended credential  
information associated with client of current  
door invocation, 280

## E

endac — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

endauclass — close audit\_class database  
file, 284, 348, 350, 352, 354, 867

endauevent — close audit\_event database  
file, 286, 359, 362, 365, 368, 371, 374, 377, 869

endauthattr — get authorization database  
entry, 145, 289, 318, 380, 383, 872

endauser — get audit\_user database  
entry, 292, 387, 389, 875

endexecattr — get execution profile entry, 294,  
322, 392, 396, 400, 503, 881

endprofattr — get profile description and  
attributes, 298, 326, 409, 413, 889

enduserattr — get user\_attr entry, 302, 330,  
420, 425, 427, 896

endutent — user accounting database  
functions, 305, 429, 432, 435, 670, 899, 1119

endutxent — user accounting database  
functions, 308, 438, 442, 446, 450, 454, 673,  
902, 1111, 1115, 1122

extract sensitivity label — getcsl, 99, 115, 391,  
878

## F

### file status

get — mldstat, mldlstat, 509, 514

### file tree

recursively descend — ftw, 333, 523

fp\_resstat — resolver routines, 266, 272, 312,  
465, 474, 679, 685, 691, 697, 703, 709, 715, 721,  
727, 733, 739, 745, 751, 757, 763, 769

free\_authattr — release memory, 145, 289, 318,  
380, 383, 872

free\_execattr — get execution profile

entry, 294, 322, 392, 396, 400, 503, 881

free\_profattr — get profile description and  
attributes, 298, 326, 409, 413, 889

free\_proflist — get execution profile entry, 294,  
298, 322, 326, 392, 396, 400, 409, 413, 503, 881,  
889

free\_userattr — get user\_attr entry, 302, 330,  
420, 425, 427, 896

ftw — walk a file tree, 333, 523

## G

generate random pronounceable password —  
randomword, 677

get execution profile entry — getexecattr, 294,  
298, 302, 322, 326, 330, 392, 396, 400, 409, 413,  
420, 425, 427, 503, 881, 889, 896

get user audit characteristics for peer —  
getpeerinfo, 405

getacdir — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

getacflg — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

getacinfo — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

getacmin — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

getacna — get audit control file  
information, 282, 338, 340, 342, 344, 346, 865

getauclassent — get audit\_class database  
entry, 284, 348, 350, 352, 354, 867

getauclassent\_r — get audit\_class database  
entry, 284, 348, 350, 352, 354, 867

getauclassnam — get audit\_class database  
entry, 284, 348, 350, 352, 354, 867

getauclassnam\_r — get audit\_class database entry, 284, 348, 350, 352, 354, 867  
 getauditflags() — generate process audit state, 404  
 getauditflagsbin — convert audit flag specifications, 356, 357, 358  
 getauditflagschar — convert audit flag specifications, 356, 357, 358  
 getauevent — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevent\_r — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevnam — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevnam\_r — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevnonam — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevnum — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauevnum\_r — get audit\_event database entry, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 getauthattr — get authorization database entry, 145, 289, 318, 380, 383, 872  
 getauthnam — get authorization database entry, 145, 289, 318, 380, 383, 872  
 getauuserent — get audit\_user database entry, 292, 387, 389, 875  
 getauuserent\_r — get audit\_user database entry, 292, 387, 389, 875  
 getauusername — get audit\_user database entry, 292, 387, 389, 875  
 getauusername\_r — get audit\_user database entry, 292, 387, 389, 875  
 getcsl — extract sensitivity label, 99, 115, 391, 878  
 getexecattr — get execution profile entry, 294, 322, 392, 396, 400, 503, 881  
 getexecprof — get execution profile entry, 294, 322, 392, 396, 400, 503, 881  
 getexecuser — get execution profile entry, 294, 322, 392, 396, 400, 503, 881  
 getpeerinfo — get peer's process characteristics, 405  
 getprofattr — get profile description and attributes, 298, 326, 409, 413, 889  
 get\_profiles — get execution profile entry, 294, 322, 392, 396, 400, 503, 881  
 getproflist — get profile description and attributes, 298, 326, 409, 413, 889  
 getprofnam — get profile description and attributes, 298, 326, 409, 413, 889  
 getuserattr — get user\_attr entry, 302, 330, 420, 425, 427, 896  
 getusername — get user\_attr entry, 302, 330, 420, 425, 427, 896  
 getuserid — get user\_attr entry, 302, 330, 420, 425, 427, 896  
 getutent — user accounting database functions, 305, 429, 432, 435, 670, 899, 1119  
 getutid — user accounting database functions, 305, 429, 432, 435, 670, 899, 1119  
 getutline — user accounting database functions, 305, 429, 432, 435, 670, 899, 1119  
 getutmp — user accounting database functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 getutmpx — user accounting database functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 getutxent — user accounting database functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 getutxid — user accounting database functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 getutxline — user accounting database functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 getvfsaent — get vfstab\_adjunct file entry, 458, 460  
 getvfsafile — search for vfstab\_adjunct file entry, 458, 460  
 grantpt — grant access to the slave pseudo-terminal device, 462  
 group IDs, supplementary  
     initialize — initgroups, 483

## H

`h_alloc` — allocate memory for a hexadecimal string, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`hextob` — hexadecimal string to binary label, 471, 480, 481, 482  
`h_free` — free hexadecimal string memory, 74, 76, 92, 94, 131, 133, 142, 463, 472  
`hostalias` — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769  
`hstrerror` — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769  
`htobcl` — hexadecimal string to binary CMW label, 471, 480, 481, 482  
`htobclear` — hexadecimal string to binary clearance, 471, 480, 481, 482  
`htobsl` — hexadecimal string to binary sensitivity label, 471, 480, 481, 482

## I

`initgroups` — initialize the supplementary group access list, 483

## K

kernel virtual memory functions  
    `kstat_read` — read or write kstat data, 484, 485  
    `kstat_write` — read or write kstat data, 484, 485  
`kstat_read` — read or write kstat data, 484, 485  
`kstat_write` — read or write kstat data, 484, 485  
`kva_match` — look up a key in a key-value array, 486

## L

label builder  
    `tsol_lbuild_create`, 487, 1091, 1096, 1101, 1106

label builder (*continued*)

`tsol_lbuild_destroy`, 487, 1091, 1096, 1101, 1106  
    `tsol_lbuild_get`, 487, 1091, 1096, 1101, 1106  
    `tsol_lbuild_set`, 487, 1091, 1096, 1101, 1106  
label clipping  
    `Xbcleartos`, 492, 1126, 1128, 1130  
    `Xbcltos`, 492, 1126, 1128, 1130  
    `Xbsltos`, 492, 1126, 1128, 1130  
label encodings file  
    get file version — `labelvers`, 496  
    information about the label encodings — `labelinfo`, 494  
label library  
    `bclearhigh`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bclearlow`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bcleartoh`, 74, 76, 92, 94, 131, 133, 142, 463, 472  
    `bcleartoh_r`, 74, 76, 92, 94, 131, 133, 142, 463, 472  
    `bcleartos`, 78, 96, 121, 135  
    `bclearundef`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bclearvalid`, 83, 125, 140  
    `bclhigh`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bcllow`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bcltobanner`, 89  
    `bcltoh`, 74, 76, 92, 94, 131, 133, 142, 463, 472  
    `bcltoh_r`, 74, 76, 92, 94, 131, 133, 142, 463, 472  
    `bcltos`, 78, 96, 121, 135  
    `bcltosl`, 99, 115, 391, 878  
    `bclundef`, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
    `bldominates`, 104, 105, 106, 107, 116  
    `blequal`, 104, 105, 106, 107, 116  
    `blinrange`, 104, 105, 106, 107, 116  
    `blinset`, 108  
    `blmaximum`, 112, 113, 114  
    `blminimum`, 112, 113, 114  
    `blstrictdom`, 104, 105, 106, 107, 116  
    `bltcolor`, 117, 119  
    `bltcolor_r`, 117, 119

label library (*continued*)

bltype, 124, 877  
bslhigh, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
bsllow, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
bsltoh, 74, 76, 92, 94, 131, 133, 142, 463, 472  
bsltoh\_r, 74, 76, 92, 94, 131, 133, 142, 463, 472  
bsltos, 78, 96, 121, 135  
bslundef, 70, 72, 81, 85, 87, 100, 110, 127, 129, 138  
bslvalid, 83, 125, 140  
getcsl, 99, 115, 391, 878  
h\_alloc, 74, 76, 92, 94, 131, 133, 142, 463, 472  
h\_free, 74, 76, 92, 94, 131, 133, 142, 463, 472  
htobcl, 471, 480, 481, 482  
htobclear, 471, 480, 481, 482  
htobsl, 471, 480, 481, 482  
labelinfo, 494  
labelvers, 496  
sbcleartos, 851, 853, 855, 857  
sbcltos, 851, 853, 855, 857  
sbsltos, 851, 853, 855, 857  
setbltype, 124, 877  
setcsl, 99, 115, 391, 878  
stobcl, 909, 912, 915, 918  
stobclear, 909, 912, 915, 918  
stobsl, 909, 912, 915, 918  
labelinfo — information about the label encodings, 494  
labelvers — label\_encodings file version, 496  
library routines for client side calls  
— clnt\_call, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_freeres, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_geterr, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_perrno, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_perror, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_sperno, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— clnt\_sperror, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820

library routines for client side calls (*continued*)

— rpc\_broadcast, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— rpc\_broadcast\_exp, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— rpc\_call, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
— rpc\_clnt\_calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820  
library routines for dealing with creation and manipulation of CLIENT handles  
— clnt\_control, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_create, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_create\_timed, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_create\_vers, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_create\_vers\_timed, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_destroy, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_dg\_create, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_pcreateerror, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_raw\_create, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_spcreateerror, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_tli\_create, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830  
— clnt\_tp\_create, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830

library routines for dealing with creation and manipulation of CLIENT handles (*continued*)

- `clnt_tp_create_timed`, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830
- `clnt_vc_create`, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830
- `rpc_clnt_create`, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830
- `rpc_createerr`, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830

library routines for RPC servers

- `rpc_svc_calls`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_dg_enablecache`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_done`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_exit`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_fdset`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_freeargs`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_getargs`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_getreq_common`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_getreq_poll`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_getreqset`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_getrpcaller`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022

library routines for RPC servers (*continued*)

- `svc_max_pollfd`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_pollfd`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_run`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022
- `svc_sendreply`, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022

`libt6` — TSIX trusted IPC library, 498

`listen` — listen for connections on a socket, 502

lock address space

- `mlockall`, 517, 521

lock memory pages

- `mlock`, 515, 519

## M

manage options for a transport endpoint —

- `t_optmgmt`, 1078

`match_execattr` — get execution profile

- entry, 294, 322, 392, 396, 400, 503, 881

memory lock or unlock

- calling process — `plock`, 664

memory management

- lock pages in memory — `mlock`, 515, 517, 519, 521

- unlock pages in memory — `munlock`, 515, 517, 519, 521

`mldgetcwd` — get pathname of current working directory, 507

`mldlstat` — get status on symbolic link file in MLD, 509, 514

`mldrealpath` — return absolute pathname, 510, 512

`mldrealpathl` — return absolute pathname, 510, 512

`mldstat` — get file status in MLD, 509, 514

## N

`nftw` — walk a file tree, 333, 523

#### NIS+ table functions

- `nis_modify_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_remove_entry`, 534, 553, 579, 601, 615, 634, 653

#### NIS+ group manipulation functions

- `nis_addmember`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_creategroup`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_destroygroup`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_groups`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_ismember`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_print_group_entry`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_removemember`, 542, 547, 550, 573, 576, 625, 642, 661
- `nis_verifygroup`, 542, 547, 550, 573, 576, 625, 642, 661

#### NIS+ log administration functions

- `nis_checkpoint`, 545, 623
- `nis_ping`, 545, 623

#### NIS+ miscellaneous functions

- `nis_freeservelist`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_freetags`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_getservlist`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_mkdir`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_rmdir`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_server`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_servstate`, 567, 569, 571, 593, 645, 647, 649, 651
- `nis_stats`, 567, 569, 571, 593, 645, 647, 649, 651

#### NIS+ namespace functions

- `nis_add`, 528, 561, 587, 595, 609, 628
- `nis_freeresult`, 528, 561, 587, 595, 609, 628
- `nis_lookup`, 528, 561, 587, 595, 609, 628
- `nis_modify`, 528, 561, 587, 595, 609, 628

#### NIS+ namespace functions (*continued*)

- `nis_names`, 528, 561, 587, 595, 609, 628
- `nis_remove`, 528, 561, 587, 595, 609, 628

#### NIS+ table functions

- `nis_add_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_first_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_list`, 534, 553, 579, 601, 615, 634, 653
- `nis_modify_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_next_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_remove_entry`, 534, 553, 579, 601, 615, 634, 653
- `nis_tables`, 534, 553, 579, 601, 615, 634, 653

## P

`plock` — lock or unlock into memory process, text, or data, 664

#### printing

translate binary CMW label to printer banner page fields — `bcltobanner`, 89

#### processes

memory lock or unlock — `plock`, 664

#### pseudo-terminal device

grant access to the slave pseudo-terminal device — `grantpt`, 462

#### pututline — user accounting database

functions, 305, 429, 432, 435, 670, 899, 1119

#### pututxline — user accounting database

functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122

## R

`randomword` — generate random pronounceable password, 677

#### read or write kstat data

— `kstat_read`, 484, 485

— `kstat_write`, 484, 485

reference sensitivity label — `bcltosl`, 99, 115, 391, 878

remote procedure calls, library routines for —  
 rpc, 775

replace sensitivity label — setcsl, 99, 115, 391, 878

res\_hostalias — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_init — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_mkquery — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nclose — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_ninit — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nmkquery — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_npquery — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nquery — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nquerydomain — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nsearch — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nsend — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_nsendsigned — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

resolver — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_query — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_querydomain — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_search — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_send — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

res\_update — resolver routines, 266, 272, 312, 465, 474, 679, 685, 691, 697, 703, 709, 715, 721, 727, 733, 739, 745, 751, 757, 763, 769

return extended credential information  
 associated with client of current door  
 invocation — door\_tcred, 280

rpc — library routines for remote procedure calls, 775

RPC bind service library routines  
 — rpcb\_getaddr, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_getallmaps, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_getmaps, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_gettime, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcbind, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_rmtcall, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_set, 784, 787, 790, 793, 796, 799, 810, 813  
 — rpcb\_unset, 784, 787, 790, 793, 796, 799, 810, 813

RPC library routines for creation and manipulation of server handles  
 — rpc\_svc\_create, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 — svc\_create, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 — svc\_destroy, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 — svc\_dg\_create, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 — svc\_fd\_create, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 — svc\_raw\_create, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038



RPC library routines for creation and manipulation of server handles (*continued*)

- `svc_tli_create`, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038
- `svc_tp_create`, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038
- `svc_vc_create`, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038

RPC library routines for registering servers

- `rpc_reg`, 836, 848, 927, 1014, 1035, 1132, 1135
- `rpc_svc_reg`, 836, 848, 927, 1014, 1035, 1132, 1135
- `svc_auth_reg`, 836, 848, 927, 1014, 1035, 1132, 1135
- `svc_reg`, 836, 848, 927, 1014, 1035, 1132, 1135
- `svc_unreg`, 836, 848, 927, 1014, 1035, 1132, 1135
- `xprt_register`, 836, 848, 927, 1014, 1035, 1132, 1135
- `xprt_unregister`, 836, 848, 927, 1014, 1035, 1132, 1135

`rpcb_getaddr` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcb_getallmaps` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcb_getmaps` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcb_gettime` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcbind` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcb_rmtcall` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpc_broadcast` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820

`rpc_broadcast_exp` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820

`rpcb_set` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpcb_unset` — library routines for RPC bind service, 784, 787, 790, 793, 796, 799, 810, 813

`rpc_call` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820

`rpc_clnt_calls` — library routines for client side calls, 148, 194, 198, 208, 212, 228, 232, 802, 806, 816, 820

`rpc_clnt_create` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830

`rpc_createerr` — library routines for dealing with creation and manipulation of CLIENT handles, 152, 158, 164, 170, 176, 182, 188, 202, 216, 222, 236, 242, 248, 254, 824, 830

`rpc_svc_calls` — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022

`rpc_svc_create` — RPC library routines for creation and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038

`rpc_svc_reg` — RPC library routines for registering servers, 836, 848, 927, 1014, 1035, 1132, 1135

## S

`sbcleartos` — binary clearance to canonical string, 851, 853, 855, 857

`sbcultos` — binary CMW label to canonical string, 851, 853, 855, 857

`sbsltos` — binary sensitivity label to canonical string, 851, 853, 855, 857

security policy, 22

`send` — send message from a socket, 859, 861, 863

`send a data unit` — `t_sndudata`, 1088

`send data or expedited data over a connection` — `t_snd`, 1084

`sendmsg` — send message from a socket, 859, 861, 863

`sendto` — send message from a socket, 859, 861, 863

`setac` — get audit control file information, 282, 338, 340, 342, 344, 346, 865

setauclass — rewind audit\_class database  
     file, 284, 348, 350, 352, 354, 867  
 setauevent — rewind audit\_event database  
     file, 286, 359, 362, 365, 368, 371, 374, 377, 869  
 setauthattr — get authorization database  
     entry, 145, 289, 318, 380, 383, 872  
 setauuser — get audit\_user database  
     entry, 292, 387, 389, 875  
 setbltype — set type of binary label, 124, 877  
 setcs1 — replace sensitivity label, 99, 115, 391, 878  
 set\_effective\_priv — set the effective privileges  
     for the current process., 879, 885, 887  
 setexecattr — get execution profile entry, 294, 322, 392, 396, 400, 503, 881  
 set\_inheritable\_priv — set the inheritable  
     privileges for the current process., 879, 885, 887  
 set\_permitted\_priv — set the permitted  
     privileges for the current process., 879, 885, 887  
 setprofattr — get profile description and  
     attributes, 298, 326, 409, 413, 889  
 setuserattr — get user\_attr entry, 302, 330, 420, 425, 427, 896  
 setutent — user accounting database  
     functions, 305, 429, 432, 435, 670, 899, 1119  
 setutxent — user accounting database  
     functions, 308, 438, 442, 446, 450, 454, 673, 902, 1111, 1115, 1122  
 socket — create an endpoint for  
     communication, 906  
 socket  
     accept a connection — accept, 38  
     bind a name — bind, 102  
     get options — getsockopt, 417, 893  
     listen for connections — listen, 502  
     send message from — send, sendto,  
         sendmsg, 859, 861, 863  
     set options — setsockopt, 417, 893  
 stobcl — string to binary CMW label, 909, 912, 915, 918  
 stobclear — string to binary clearance, 909, 912, 915, 918  
 stobsl — string to binary sensitivity label, 909, 912, 915, 918

## STREAMS

accept a connection on a socket —  
     accept, 38  
 bind a name to a socket — bind, 102  
 create an endpoint for communication —  
     socket, 906  
 get and set socket options — getsockopt,  
     setsockopt, 417, 893  
 listen for connections on a socket —  
     listen, 502  
 send a message from a socket — send,  
     sendto, sendmsg, 859, 861, 863  
 svc\_auth\_reg — RPC library routines for  
     registering servers, 836, 848, 927, 1014, 1035, 1132, 1135  
 svc\_control — RPC library routines for creation  
     and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_create — RPC library routines for creation  
     and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_destroy — RPC library routines for creation  
     and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_dg\_create — RPC library routines for  
     creation and manipulation of server  
     handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_dg\_enablecache — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_done — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_exit — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_fdset — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_freeargs — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_getargs — library routines for RPC  
     servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022

svc\_getreq\_common — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_getreq\_poll — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_getreqset — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_getrpcaller — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_max\_pollfd — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_pollfd — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_reg — RPC library routines for registering servers, 836, 848, 927, 1014, 1035, 1132, 1135  
 svc\_run — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_sendreply — library routines for RPC servers, 839, 946, 951, 956, 965, 970, 975, 980, 985, 990, 995, 1000, 1005, 1017, 1022  
 svc\_tli\_create — RPC library routines for creation and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_tp\_create — RPC library routines for creation and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038  
 svc\_vc\_create — RPC library routines for creation and manipulation of server handles, 844, 930, 934, 938, 942, 961, 1010, 1027, 1031, 1038

**T**  
 t6alloc\_blk — allocates security-attribute control structure and buffer, 1042, 1049  
 t6attr\_query — get mask indicating which attributes came from templates, 1043  
 t6clear\_blk — clear security attributes, 1044  
 t6cmp\_blk — compare security attributes, 1045  
 t6copy\_blk — copy security attributes, 1046  
 t6dup\_blk — duplicate security attributes, 1047  
 t6ext\_attr — manipulate network-endpoint security options, 1048, 1057  
 t6free\_blk — frees security-attribute control structure and buffer, 1042, 1049  
 t6get\_attr — get security attributes from the security-attribute buffer handled by a control structure, 1050, 1063  
 t6get\_endpt\_default — get endpoint default attributes, 1052, 1054, 1065, 1067  
 t6get\_endpt\_mask — get endpoint mask, 1052, 1054, 1065, 1067  
 t6last\_attr — examine the security attributes on the previous byte of data, 1056, 1058  
 t6new\_attr — manipulate network-endpoint security options, 1048, 1057  
 t6peek\_attr — examine the security attributes on the next byte of data, 1056, 1058  
 t6recvfrom — read security attributes and data from a trusted endpoint, 1059  
 t6sendto — specify security attributes to send with data on a trusted endpoint, 1061  
 t6set\_attr — set security attributes in the security-attribute buffer handled by a control structure, 1050, 1063  
 t6set\_endpt\_default — set endpoint default attributes, 1052, 1054, 1065, 1067  
 t6set\_endpt\_mask — set endpoint mask, 1052, 1054, 1065, 1067  
 t6size\_attr — get the size of a particular attribute from the control structure, 1069  
 t\_accept — accept a connection request, 1070  
 t\_bind — bind an address to a transport endpoint, 1074  
 terminal device, slave pseudo  
   grant access — grantpt, 462  
 t\_optmgmt — manage options for a transport endpoint, 1078  
 translate binary clearance to string —  
   bclear\_tos, 78, 96, 121, 135  
 translate binary CMW label to string —  
   bcl\_tos, 78, 96, 121, 135

translate binary label to printer banner —  
 bcltobanner, 89  
 translate binary sensitivity label to string —  
 bsltos, 78, 96, 121, 135  
 TSIX trusted IPC library — libt6, 498  
 t\_snd — send data or expedited data over a  
 connection, 1084  
 t\_sndudata — send a data unit, 1088  
 tsol\_lbuild\_create — create a user interface for  
 interactively building a valid label or  
 clearance, 487, 1091, 1096, 1101, 1106  
 tsol\_lbuild\_destroy — destroy label builder user  
 interface, 487, 1091, 1096, 1101, 1106  
 tsol\_lbuild\_get — get user interface for label  
 builder, 487, 1091, 1096, 1101, 1106  
 tsol\_lbuild\_set — change user interface  
 information for label builder, 487, 1091,  
 1096, 1101, 1106

## U

unlock address space  
 — munlockall, 517, 521  
 unlock memory pages  
 — munlock, 515, 519  
 updwtmp — user accounting database  
 functions, 308, 438, 442, 446, 450, 454, 673,  
 902, 1111, 1115, 1122  
 updwtmpx — user accounting database  
 functions, 308, 438, 442, 446, 450, 454, 673,  
 902, 1111, 1115, 1122  
 user accounting database functions —  
 getutent, 305, 308, 429, 432, 435, 438, 442,  
 446, 450, 454, 670, 673, 899, 902, 1111, 1115,  
 1119, 1122  
 utmpname — user accounting database  
 functions, 305, 429, 432, 435, 670, 899, 1119  
 utmpxname — user accounting database  
 functions, 308, 438, 442, 446, 450, 454, 673,  
 902, 1111, 1115, 1122

## V

vfstab\_adjunct file  
 get entry — getvfsaent, 458, 460

vfstab\_adjunct file (*continued*)  
 search for entry — getvfsafile, 458, 460

## W

write user-level audit records — auditwrite, 42

## X

Xbclearaos — binary clearance to string with  
 font list, 492, 1126, 1128, 1130  
 Xbcltos — binary CMW label to string with font  
 list, 492, 1126, 1128, 1130  
 Xbsltos — binary sensitivity label to string with  
 font list, 492, 1126, 1128, 1130  
 XTSOLgetClientAttributes — get client  
 attrs, 1138  
 XTSOLgetPropAttributes — get all prop  
 attrs, 1139  
 XTSOLgetPropLabel — set resource label, 1140  
 XTSOLgetPropUID — get property uid, 1141  
 XTSOLgetResAttributes — get all attrs, 1142  
 XTSOLgetResLabel — get resource label, 1143  
 XTSOLgetResUID — set resource uid, 1144  
 XTSOLgetSSHeight — get screen stripe  
 height, 1145  
 XTSOLgetWorkstationOwner — get  
 ownership, 1146  
 XTSOLisWindowTrusted — test Trusted  
 Window, 1147  
 XTSOLMakeTPWindow — Make Trusted path  
 window, 1148  
 XTSOLsetPolyInstInfo — set poly instantiation  
 info, 1149  
 XTSOLsetPropLabel — set resource label, 1150  
 XTSOLsetPropUID — set property uid, 1151  
 XTSOLsetResLabel — set resource label, 1152  
 XTSOLsetResUID — set resource uid, 1153  
 XTSOLsetSessionHI — set SL, 1154  
 XTSOLsetSessionLO — set SL, 1155  
 XTSOLsetSSHeight — set screen stripe  
 height, 1156  
 XTSOLsetWorkstationOwner — set  
 ownership, 1157  
 XTSOLShutdown — shutdown system, 1158