

# man pages section 3: Networking Library Functions

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

Part No: 816–0214–06 December 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'APITITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.





011030@2471

## Contents

Preface 11

**Networking Library Functions** 17 accept(3SOCKET) 18 accept(3XNET) 20 ber\_decode(3LDAP) 22 ber\_encode(3LDAP) 27 bind(3SOCKET) 31 bind(3XNET) 33 byteorder(3SOCKET) 35 connect(3SOCKET) 36 connect(3XNET) 39 dial(3NSL) 43 doconfig(3NSL) 45 endhostent(3XNET) 47 endnetent(3XNET) 49 endprotoent(3XNET) 51 endservent(3XNET) 53 ethers(3SOCKET) 55 fn\_attr\_bind(3XFN) 57 fn\_attr\_create\_subcontext(3XFN) 58 fn\_attr\_ext\_search(3XFN) 59 fn\_attr\_get(3XFN) 66 fn\_attr\_get\_ids(3XFN) 67 fn\_attr\_get\_values(3XFN) 68

FN\_attribute\_t(3XFN) 71 fn\_attr\_modify(3XFN) 73 FN\_attrmodlist\_t(3XFN) 75 fn\_attr\_multi\_get(3XFN) 78 fn\_attr\_multi\_modify(3XFN) 82 fn\_attr\_search(3XFN) 84 FN\_attrset\_t(3XFN) 89 FN\_attrvalue\_t(3XFN) 91 FN\_composite\_name\_t(3XFN) 92 97 FN\_compound\_name\_t(3XFN) fn\_ctx\_bind(3XFN) 102 fn\_ctx\_create\_subcontext(3XFN) 104 fn\_ctx\_destroy\_subcontext(3XFN) 105 fn\_ctx\_equivalent\_name(3XFN) 106 fn\_ctx\_get\_ref(3XFN) 108 fn\_ctx\_get\_syntax\_attrs(3XFN) 109 fn\_ctx\_handle\_destroy(3XFN) 111 fn\_ctx\_handle\_from\_initial(3XFN) 112 fn\_ctx\_handle\_from\_ref(3XFN) 114 fn\_ctx\_list\_bindings(3XFN) 116 fn\_ctx\_list\_names(3XFN) 117 fn\_ctx\_lookup(3XFN) 120 fn\_ctx\_lookup\_link(3XFN) 121 fn\_ctx\_rename(3XFN) 122 FN\_ctx\_t(3XFN) 124 fn\_ctx\_unbind(3XFN) 127 FN\_identifier\_t(3XFN) 128 FN\_ref\_addr\_t(3XFN) 129 FN\_ref\_t(3XFN) 131 FN\_search\_control\_t(3XFN) 134 FN\_search\_filter\_t(3XFN) 137 FN\_status\_t(3XFN) 144 FN\_string\_t(3XFN) 149 getaddrinfo(3SOCKET) 153 gethostbyname(3NSL) 157 gethostname(3XNET) 163 getipnodebyname(3SOCKET) 164

4 man pages section 3: Networking Library Functions • December 2001 (Beta)

getnetbyname(3SOCKET) 170 getnetconfig(3NSL) 173 getnetpath(3NSL) 175 getpeername(3SOCKET) 177 getpeername(3XNET) 178 getprotobyname(3SOCKET) 180 getpublickey(3NSL) 183 getrpcbyname(3NSL) 184 getservbyname(3SOCKET) 187 getsockname(3SOCKET) 191 getsockname(3XNET) 192 getsockopt(3SOCKET) 193 getsockopt(3XNET) 197 gss\_accept\_sec\_context(3GSS) 200 gss\_acquire\_cred(3GSS) 206 gss\_add\_cred(3GSS) 209 gss\_add\_oid\_set\_member(3GSS) 213 214 gss\_canonicalize\_name(3GSS) gss\_compare\_name(3GSS) 216 gss\_context\_time(3GSS) 217 gss\_create\_empty\_oid\_set(3GSS) 218 gss\_delete\_sec\_context(3GSS) 219 gss\_display\_name(3GSS) 221 gss\_display\_status(3GSS) 223 gss\_duplicate\_name(3GSS) 225 gss\_export\_name(3GSS) 226 gss\_export\_sec\_context(3GSS) 227 gss\_get\_mic(3GSS) 229 gss\_import\_name(3GSS) 231 gss\_import\_sec\_context(3GSS) 233 235 gss\_indicate\_mechs(3GSS) gss\_init\_sec\_context(3GSS) 236 243 gss\_inquire\_context(3GSS) 246 gss\_inquire\_cred(3GSS) gss\_inquire\_cred\_by\_mech(3GSS) 248 gss\_inquire\_mechs\_for\_name(3GSS) 250 gss\_inquire\_names\_for\_mech(3GSS) 252 gss\_oid\_to\_str(3GSS) 253 gss\_process\_context\_token(3GSS) 255 gss\_release\_buffer(3GSS) 257 gss\_release\_cred(3GSS) 258 gss\_release\_name(3GSS) 259 gss\_release\_oid(3GSS) 260 gss\_release\_oid\_set(3GSS) 261 gss\_str\_to\_oid(3GSS) 262 gss\_test\_oid\_set\_member(3GSS) 264 gss\_unwrap(3GSS) 265 gss\_verify\_mic(3GSS) 267 gss\_wrap(3GSS) 269 gss\_wrap\_size\_limit(3GSS) 271 htonl(3XNET) 273 if\_nametoindex(3NSL) 274 if\_nametoindex(3XNET) 276 278 inet(3SOCKET) inet\_addr(3XNET) 282 ldap(3LDAP) 284 ldap\_abandon(3LDAP) 295 ldap\_add(3LDAP) 296 ldap\_ber\_free(3LDAP) 298 299 ldap\_bind(3LDAP) ldap\_charset(3LDAP) 302 304 ldap\_compare(3LDAP) ldap\_control\_free(3LDAP) 306 ldap\_delete(3LDAP) 307 309 ldap\_disptmpl(3LDAP) ldap\_entry2text(3LDAP) 315 ldap\_error(3LDAP) 318 ldap\_first\_attribute(3LDAP) 322 ldap\_first\_entry(3LDAP) 324 ldap\_first\_message(3LDAP) 326 ldap\_friendly(3LDAP) 327 ldap\_get\_dn(3LDAP) 328 ldap\_get\_entry\_controls(3LDAP) 330 ldap\_getfilter(3LDAP) 331

6 man pages section 3: Networking Library Functions • December 2001 (Beta)

ldap\_get\_lang\_values(3LDAP) 333 ldap\_get\_option(3LDAP) 335 ldap\_get\_values(3LDAP) 338 ldap\_memcache(3LDAP) 340 ldap\_modify(3LDAP) 343 ldap\_modrdn(3LDAP) 345 ldap\_open(3LDAP) 347 ldap\_parse\_result(3LDAP) 349 ldap\_result(3LDAP) 350 ldap\_search(3LDAP) 352 ldap\_searchprefs(3LDAP) 354 ldap\_sort(3LDAP) 356 ldap\_ufn(3LDAP) 358 ldap\_url(3LDAP) 360 ldap\_version(3LDAP) 363 listen(3SOCKET) 364 listen(3XNET) 365 netdir(3NSL) 367 nis\_error(3NSL) 371 nis\_groups(3NSL) 372 nis\_local\_names(3NSL) 375 nis\_names(3NSL) 377 nis\_objects(3NSL) 383 nis\_ping(3NSL) 392 nis\_server(3NSL) 393 nis\_subr(3NSL) 395 nis\_tables(3NSL) 398 nlsgetcall(3NSL) 406 nlsprovider(3NSL) 407 nlsrequest(3NSL) 408 rcmd(3SOCKET) 410 recv(3SOCKET) 412 414 recv(3XNET) recvfrom(3XNET) 417 recvmsg(3XNET) 420 resolver(3RESOLV) 423 rexec(3SOCKET) 429

rpc(3NSL) 431 rpcbind(3NSL) 440 rpc\_clnt\_auth(3NSL) 442 rpc\_clnt\_calls(3NSL) 444 rpc\_clnt\_create(3NSL) 448rpc\_control(3NSL) 455 rpc\_gss\_getcred(3NSL) 457 rpc\_gss\_get\_error(3NSL) 459 rpc\_gss\_get\_mechanisms(3NSL) 460 rpc\_gss\_get\_principal\_name(3NSL) 462 rpc\_gss\_max\_data\_length(3NSL) 464 rpc\_gss\_mech\_to\_oid(3NSL) 465 rpc\_gss\_seccreate(3NSL) 467 rpc\_gss\_set\_callback(3NSL) 469 rpc\_gss\_set\_defaults(3NSL) 471 rpc\_gss\_set\_svc\_name(3NSL) 472 474 rpc\_rac(3RAC) rpcsec\_gss(3NSL) 478 rpc\_soc(3NSL) 483 rpc\_svc\_calls(3NSL) 495 rpc\_svc\_create(3NSL) 499 rpc\_svc\_err(3NSL) 504 rpc\_svc\_input(3NSL) 506 rpc\_svc\_reg(3NSL) 508 rpc\_xdr(3NSL) 510 rstat(3RPC) 512 rusers(3RPC) 513 rwall(3RPC) 514 secure\_rpc(3NSL) 515 send(3SOCKET) 519 send(3XNET) 521 sendmsg(3XNET) 524 sendto(3XNET) 528 setsockopt(3XNET) 532 shutdown(3SOCKET) 535 shutdown(3XNET) 536 slp\_api(3SLP) 537

8 man pages section 3: Networking Library Functions • December 2001 (Beta)

SLPClose(3SLP) 547 SLPDelAttrs(3SLP) 548 SLPDereg(3SLP) 550 SLPEscape(3SLP) 552 SLPFindAttrs(3SLP) 554 SLPFindScopes(3SLP) 556 SLPFindSrvs(3SLP) 558 SLPFindSrvTypes(3SLP) 560 SLPFree(3SLP) 562 SLPGetProperty(3SLP) 563 SLPGetRefreshInterval(3SLP) 564 565 SLPOpen(3SLP) SLPParseSrvURL(3SLP) 567 SLPReg(3SLP) 569 SLPSetProperty(3SLP) 571 slp\_strerror(3SLP) 572 SLPUnescape(3SLP) 573 socket(3SOCKET) 575 socket(3XNET) 578 socketpair(3SOCKET) 580 socketpair(3XNET) 581 583 spray(3SOCKET) t\_accept(3NSL) 585 t\_alloc(3NSL) 589 t\_bind(3NSL) 592 t\_close(3NSL) 596 t\_connect(3NSL) 598 t\_errno(3NSL) 602 t\_error(3NSL) 604 t\_free(3NSL) 606 t\_getinfo(3NSL) 608 t\_getprotaddr(3NSL) 612 t\_getstate(3NSL) 614 t\_listen(3NSL) 616 t\_look(3NSL) 619 t\_open(3NSL) 621 t\_optmgmt(3NSL) 625

t\_rcv(3NSL) 633 t\_rcvconnect(3NSL) 636 639 t\_rcvdis(3NSL) t\_rcvrel(3NSL) 641 t\_rcvreldata(3NSL) 643 t\_rcvudata(3NSL) 645 t\_rcvuderr(3NSL) 648 t\_rcvv(3NSL) 650 t\_rcvvudata(3NSL) 653 t\_snd(3NSL) 655 t\_snddis(3NSL) 659 t\_sndrel(3NSL) 661 t\_sndreldata(3NSL) 663 t\_sndudata(3NSL) 665 t\_sndv(3NSL) 668 t\_sndvudata(3NSL) 672 t\_strerror(3NSL) 675 t\_sync(3NSL) 676 t\_sysconf(3NSL) 678 t\_unbind(3NSL) 679 xdr(3NSL) 681 xdr\_admin(3NSL) 683 xdr\_complex(3NSL) 685 xdr\_create(3NSL) 688 xdr\_simple(3NSL) 690 xfn(3XFN) 694 xfn\_attributes(3XFN) 695 xfn\_composite\_names(3XFN) 698 xfn\_compound\_names(3XFN) 699 xfn\_links(3XFN) 702 xfn\_status\_codes(3XFN) 704 ypclnt(3NSL) 708 yp\_update(3NSL) 713

Index 715

### Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. 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.

### Overview

The following contains a brief description of each man page section 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.
- 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 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 can 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 command functions documented, followed by a brief description of what they do.	
SYNOPSIS	This section shows the syntax of commands on functions. When a command or file does not ex- in the standard path, its full path name is show Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument or is required.	
	The follosection:	owing special characters are used in this
	[]	Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
		Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename ".
	I	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, and functions are described under USAGE.
IOCTL	This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mtio(7I).
OPTIONS	This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or $-1$ , these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than

	one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:
	Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as example%, or if the user must be superuser, example%, or if the user must be superuser, 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.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.

14 man pages section 3: Networking Library Functions • December 2001 (Beta)

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.

16 man pages section 3: Networking Library Functions • December 2001 (Beta)

Networking Library Functions

accept(3SOCKET)		
NAME	accept – accept a connection on a socket	
SYNOPSIS	<pre>cc [ flag ] filelsoc #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>	ket -lnsl [ <i>library</i> ]
	<pre>int accept(int s, struc</pre>	t sockaddr * <i>addr</i> , socklen_t * <i>addrlen</i> );
DESCRIPTION		
	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.	
	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.	
	The accept() function is used with connection-based socket types, currently with SOCK_STREAM.	
		r poll(2) a socket for the purpose of an accept() by d. However, this will only indicate when a connect l necessary to call accept().
RETURN VALUES	The accept() function return integer that is a descriptor for	ns –1 on error. If it succeeds, it returns a non-negative the accepted socket.
ERRORS	accept() will fail if:	
	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 netconfig file.

18 man pages section 3: Networking Library Functions • Last Revised 16 May 1997

				accept(SSOCKE
	ENOMEM	There was ir complete the	sufficient user memory e operation.	v available to
	ENOSR		nsufficient STREAMS r the operation.	esources available
	ENOTSOCK	The descript	or does not reference a	socket.
	EOPNOTSUPP	The reference	ed socket is not of type	SOCK_STREAM.
	EPROTO	STREAMS p	rror has occurred; for e rotocol stack has not b as already been release	een initialized or th
	EWOULDBLOCK		s marked as non-blocki are present to be accep	
	See attributes(5) for descr	riptions of the	following attributes:	
	ATTRIBUTE TYPE		ATTRIBUTE	EVALUE
)	MT-Level poll(2), bind(3SOCKET), cd socket(3SOCKET), netcon			
o	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
50	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
50	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
O	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
0	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
0	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
O	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
0	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	
0	poll(2), bind(3SOCKET), co		KET),listen(3SOCK)	

#### accept(3XNET)

NAME	accept – accept a new connection on a socket	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>	
	<pre>int accept(int socket, struct sockaddr *address, socklen_t     *address_len);</pre>	
DESCRIPTION	The accept() function extracts the first connection on the queue of pending connections, creates a new socket with the same socket type protocol and address family as the specified socket, and allocates a new file descriptor for that socket.	
	The function takes	s the following arguments:
	socket	Specifies a socket that was created with socket(3XNET), has been bound to an address with bind(3XNET), and has issued a successful call to listen(3XNET).
	address	Either a null pointer, or a pointer to a sockaddr structure where the address of the connecting socket will be returned.
	address_len	Points to a socklen_t which on input specifies the length of the supplied sockaddr structure, and on output specifies the length of the stored address.
	stored in the sock	null pointer, the address of the peer for the accepted connection is caddr structure pointed to by <i>address</i> , and the length of this address ject pointed to by <i>address_len</i> .
		h of the address is greater than the length of the supplied sockaddr ed address will be truncated.
		mits connections by unbound clients, and the peer is not bound, red in the object pointed to by <i>address</i> is unspecified.
	the file descriptor the listen(3XNE	is empty of connection requests and O_NONBLOCK is not set on for the socket, accept () will block until a connection is present. If ET) queue is empty of connection requests and O_NONBLOCK is set tor for the socket, accept () will fail and set errno to EAGAIN or
		et cannot itself accept more connections. The original socket remains ept more connections.
USAGE	When a connection the socket is ready	n is available, select(3C) will indicate that the file descriptor for 7 for reading.
<b>RETURN VALUES</b>		ompletion, accept() returns the nonnegative file descriptor of the Otherwise, -1 is returned and errno is set to indicate the error.
ERRORS	The accept() function will fail if:	

20 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

	EAGAIN	
	EWOULDBLOCK	O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.
	EBADF	The socket argument is not a valid file descriptor.
	ECONNABORTED	A connection has been aborted.
	EFAULT	The <i>address</i> or <i>address_len</i> parameter can not be accessed or written.
	EINTR	The accept() function was interrupted by a signal that was caught before a valid connection arrived.
	EINVAL	The <i>socket</i> is not accepting connections.
	EMFILE	OPEN_MAX file descriptors are currently open in the calling process.
	ENFILE	The maximum number of file descriptors in the system are already open.
	ENOTSOCK	The socket argument does not refer to a socket.
	EOPNOTSUPP	The socket type of the specified socket does not support accepting connections.
	The accept () function may	fail if:
	ENOBUFS	No buffer space is available.
	ENOMEM	There was insufficient memory available to complete the operation.
	ENOSR	There was insufficient STREAMS resources available to complete the operation.
	EPROTO	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO bind(3XNET), connect(3XNET), listen(3XNET), socket(3XNET), attributes(5)

Networking Library Functions 21

#### ber\_decode(3LDAP)

NAME	ber_decode, ber_alloc_t, ber_free, ber_bvdup, ber_init, ber_flatten, ber_get_next, ber_skiptag, ber_peek_tag, ber_scanf, ber_get_int, ber_get_stringa, ber_get_stringal, ber_get_stringb, ber_get_null, ber_get_boolean, ber_get_bitstring, ber_first_element, ber_next_element, ber_bvfree, ber_bvecfree – Basic Encoding Rules library decoding functions
SYNOPSIS	cc[flag] filelldap[ library]
	<pre>#include <lber.h></lber.h></pre>
	<pre>BerElement *ber_alloc_t(int options);</pre>
	struct berval <b>*ber_bvdup</b> (struct berval $*bv$ );
	<pre>void ber_free(BerElement *ber, int freebuf);</pre>
	<pre>BerElement *ber_init(struct berval *bv);</pre>
	<pre>int ber_flatten(BerElement *ber, struct berval **bvPtr);</pre>
	<pre>ber_get_next(Sockbuf *sb, unsigned long *len, char *bv_val);</pre>
	<pre>ber_skip_tag(BerElement **ber, unsigned long **len);</pre>
	<pre>ber_peek_tag(BerElement **ber, unsigned long **len);</pre>
	<pre>ber_get_int(BerElement **ber, long **num);</pre>
	<pre>ber_get_stringb(BerElement **ber, char **buf, unsigned long **len);</pre>
	<pre>ber_get_stringa(BerElement **ber, char ***buf);</pre>
	<pre>ber_get_stringal(BerElement **ber, struct berval ***bv);</pre>
	<pre>ber_get_null(BerElement **ber);</pre>
	<pre>ber_get_boolean(BerElement **ber, int **bool);</pre>
	<pre>ber_get_bitstringa(BerElement **ber, char ***buf, unsigned long</pre>
	<pre>ber_first_element(BerElement **ber, unsigned long **len, char</pre>
	<pre>ber_next_element(BerElement **ber, unsigned long **len, char **cookie);</pre>
	<pre>ber_scanf(BerElement **ber, char **fmt [, arg]);</pre>
	<pre>ber_bvfree(struct berval **bv);</pre>
	<pre>ber_bvecfree(struct berval ***bvec);</pre>
DESCRIPTION	These functions provide a subfunction interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these functions support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. In addition, these lightweight BER functions restrict tags

22 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

and class to fit in a single octet (this means the actual tag must be less than 31). When a "tag" is specified in the descriptions below, it refers to the tag, class, and primitive or constructed bit in the first octet of the encoding. This man page describes the decoding functions in the lber library. See ber\_encode(3LDAP) for details on the corresponding encoding functions.

Normally, the only functions that need be called by an application are ber\_get\_next() to get the next BER element and ber\_scanf() to do the actual decoding. In some cases, ber\_peek\_tag() may also need to be called in normal usage. The other functions are provided for those applications that need more control than ber\_scanf() provides. In general, these functions return the tag of the element decoded, or -1 if an error occurred.

The ber\_get\_next() function is used to read the next BER element from the given Sockbuf, *sb*. A Sockbuf consists of the descriptor (usually socket, but a file descriptor works just as well) from which to read, and a BerElement structure used to maintain a buffer. On the first call, the *sb\_ber* struct should be zeroed. It strips off and returns the leading tag byte, strips off and returns the length of the entire element in *len*, and sets up *ber* for subsequent calls to ber scanf(), and all to decode the element.

The ber\_scanf() function is used to decode a BER element in much the same way that scanf(3C) works. It reads from *ber*, a pointer to a BerElement such as returned by ber\_get\_next(), interprets the bytes according to the format string fmt, and stores the results in its additional arguments. The format string contains conversion specifications which are used to direct the interpretation of the BER element. The format string can contain the following characters.

-a	Octet string. A char ** should be supplied. Memory is allocated, filled with the contents of the octet string, null-terminated, and returned in the parameter.
- S	Octet string. A char * buffer should be supplied, followed by a pointer to an integer initialized to the size of the buffer. Upon return, the null-terminated octet string is put into the buffer, and the integer is set to the actual size of the octet string.
-0	Octet string. A struct ber_val ** should be supplied, which upon return points to a memory allocated struct berval containing the octet string and its length. ber_bvfree() can be called to free the allocated memory.
-b	Boolean. A pointer to an integer should be supplied.
-i	Integer. A pointer to an integer should be supplied.
-B	Bitstring. A char ** should be supplied which will point to the memory allocated bits, followed by an unsigned long *, which will point to the length (in bits) of the bitstring returned.
-n	Null. No parameter is required. The element is simply skipped if it is recognized.

#### ber\_decode(3LDAP)

-v	Sequence of octet strings. A char *** should be supplied, which upon return points to a memory allocated null-terminated array of char *'s containing the octet strings. NULL is returned if the sequence is empty.
-V	Sequence of octet strings with lengths. A struct berval *** should be supplied, which upon return points to a memory allocated, null-terminated array of struct berval *'s containing the octet strings and their lengths. NULL is returned if the sequence is empty. ber_bvecfree() can be called to free the allocated memory.
-x	Skip element. The next element is skipped.
- {	Begin sequence. No parameter is required. The initial sequence tag and length are skipped.
- }	End sequence. No parameter is required and no action is taken.
- ]&	Begin set. No parameter is required. The initial set tag and length are skipped.
-]	End set. No parameter is required and no action is taken.
	t () function tries to interpret the next element as an integer, It in <i>num</i> . The tag of whatever it finds is returned on success, –1 on
buffer. The <i>len</i> para the length of the o	ringb() function is used to read an octet string into a preallocated ameter should be initialized to the size of the buffer, and will contain ctet string read upon return. The buffer should be big enough to g value plus a terminating NULL byte.
The ber_get_st: octet string is read	ringa() function is used to allocate memory space into which an
	ringal() function is used to allocate memory space into which an being have read. It takes a struct berval **, and returns the result in
The ber_get_nu: the element it skip	ll() function is used to read a NULL element. It returns the tag of s over.
	olean() function is used to read a boolean value. It is called theget_int() is called.
char ** which will	tstringa() function is used to read a bitstring value. It takes a hold the allocated memory bits, followed by an unsigned long *, o the length (in bits) of the bitstring returned.

24 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

The ber\_first\_element() function is used to return the tag and length of the first element in a set or sequence. It also returns in *cookie* a magic cookie parameter that should be passed to subsequent calls to ber\_next\_element(), which returns similar information.

ber\_alloc\_t() constructs and returns BerElement. A null pointer is returned on error. The options field contains a bitwise-or of options which are to be used when generating the encoding of this BerElement. One option is defined and must always be supplied:

#define LBER USE DER 0x01

When this option is present, lengths will always be encoded in the minimum number of octets. Note that this option does not cause values of sets and sequences to be rearranged in tag and byte order, so these functions are not suitable for generating DER output as defined in X.509 and X.680

The ber\_init function constructs a BerElement and returns a new BerElement containing a copy of the data in the bv argument. ber\_init returns the null pointer on error.

ber\_free() frees a BerElement which is returned from the API calls ber\_alloc\_t() or ber\_init(). Each BerElement must be freed by the caller. The second argument *freebuf* should always be set to 1 to ensure that the internal buffer used by the BER functions is freed as well as the BerElement container itself.

ber\_bvdup() returns a copy of a *berval*. The *bv\_val* field in the returned *berval* points to a different area of memory as the *bv\_val* field in the argument *berval*. The null pointer is returned on error (that is, is out of memory).

The ber\_flatten routine allocates a struct berval whose contents are BER encoding taken from the *ber* argument. The *bvPtr* pointer points to the returned *berval*, which must be freed using ber\_bvfree(). This routine returns 0 on success and -1 on error.

EXAMPLES

**EXAMPLE 1** Assume the variable *ber* contains a lightweight BER encoding of the following ASN.1 object:

```
AlmostASearchRequest := SEQUENCE {
   baseObject DistinguishedName,
   scope ENUMERATED {
     baseObject (0),
     singleLevel (1),
     wholeSubtree (2)
   },
   derefAliases ENUMERATED {
     neverDerefaliases (0),
     derefInSearching (1),
     derefFindingBaseObj (2),
     alwaysDerefAliases (3N)
   },
   sizelimit INTEGER (0 .. 65535),
   timelimit INTEGER (0 .. 65535),
```

Networking Library Functions 25

ber\_decode(3LDAP)

**EXAMPLE 1** Assume the variable *ber* contains a lightweight BER encoding of the following ASN.1 object: (Continued) attrsOnly BOOLEAN, attributes SEQUENCE OF AttributeType } **EXAMPLE 2** The element can be decoded using ber scanf() as follows. scope, ali, size, time, attrsonly; int char \*dn, \*\*attrs; if ( ber\_scanf( ber, "{aiiiib{v}}", &dn, &scope, &ali, &size, &time, &attrsonly, &attrs ) == -1 ) /\* error \*/ else /\* success \*/ If an error occurs during decoding, generally these functions return –1. ERRORS NOTES The return values for all of these functions are declared in the <lber.h> header file. Some functions may allocate memory which must be freed by the calling application. **ATTRIBUTES** See attributes(5) for a description of the following attributes: ATTRIBUTE TYPE ATTRIBUTE VALUE Availability SUNWIIdap (32-bit) SUNWldapx (64-bit) Stability Level Evolving SEE ALSO ber encode(3LDAP) Yeong, W., Howes, T., and Hardcastle-Kille, S., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992. Information Processing - Open Systems Interconnection - Model and Notation -Service Definition - Specification of Basic Encoding Rules for Abstract Syntax Notation One, International Organization for Standardization, International Standard 8825.

26 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

NAME	<pre>ber_encode, ber_alloc, ber_printf, ber_put_int, ber_put_ostring, ber_put_string, ber_put_null, ber_put_boolean, ber_put_bitstring, ber_start_seq, ber_start_set, ber_put_seq, ber_put_set - simplified Basic Encoding Rules library encoding functions</pre>
SYNOPSIS	cc[ flag ] filelldap[ library ]
	<pre>#include <lber.h></lber.h></pre>
	<pre>BerElement*ber_alloc();</pre>
	<pre>ber_printf(BerElement *ber, char **fmt[, arg]);</pre>
	<pre>ber_put_int(BerElement *ber, long num, char tag);</pre>
	<pre>ber_put_ostring(BerElement *ber, char **str, unsigned long len, char</pre>
	<pre>ber_put_string(BerElement *ber, char **str, char tag);</pre>
	<pre>ber_put_null(BerElement *ber, char tag);</pre>
	<pre>ber_put_boolean(BerElement *ber, int bool, char tag);</pre>
	<pre>ber_put_bitstring(BerElement *ber, char *str, int blen, char tag);</pre>
	<pre>ber_start_seq(BerElement *ber, char tag);</pre>
	<pre>ber_start_set(BerElement *ber, char tag);</pre>
	<pre>ber_put_seq(BerElement *ber);</pre>
	<pre>ber_put_set(BerElement *ber);</pre>
DESCRIPTION	These functions provide a subfunction interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these functions support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. In addition, these lightweight BER functions restrict tags and class to fit in a single octet (this means the actual tag must be less than 31). When a "tag" is specified in the descriptions below, it refers to the tag, class, and primitive or constructed bit in the first octet of the encoding. This man page describes the encoding functions in the lber library. See ber_decode(3LDAP) for details on the corresponding decoding functions.
	Normally, the only functions that need be called by an application are ber_alloc(), to allocate a BER element, and ber_printf() to do the actual encoding. The other functions are provided for those applications that need more control than ber_printf() provides. In general, these functions return the length of the element encoded, or -1 if an error occurred.
	The ber_alloc() function is used to allocate a new BER element.
	The ber_printf() function is used to encode a BER element in much the same way that sprintf(3S) works. One important difference, though, is that some state

#### ber\_encode(3LDAP)

information is kept with the <i>ber</i> parameter so that multiple calls can be made to ber_printf() to append things to the end of the BER element. Ber_printf() writes to <i>ber</i> , a pointer to a BerElement such as returned by ber_alloc(). It interprets and formats its arguments according to the format string fmt. The format string can contain the following characters:		
-b	Boolean. An integer parameter should be supplied. A boolean element is output.	
-i	Integer. An integer parameter should be supplied. An integer element is output.	
-B	Bitstring. A char * pointer to the start of the bitstring is supplied, followed by the number of bits in the bitstring. A bitstring element is output.	
-n	Null. No parameter is required. A null element is output.	
-0	Octet string. A char * is supplied, followed by the length of the string pointed to. An octet string element is output.	
- 5	- s Octet string. A null-terminated string is supplied. An octet string element is output, not including the trailing NULL octet.	
-t	t Tag. An int specifying the tag to give the next element is provided. This works across calls.	
-v	Several octet strings. A null-terminated array of char *'s is supplied. Note that a construct like '{v}' is required to get an actual SEQUENCE OF octet strings.	
-{	Begin sequence. No parameter is required.	
-}	End sequence. No parameter is required.	
- ] &	Begin set. No parameter is required.	
-]	End set. No parameter is required.	
The ber_put_int() function writes the integer element <i>num</i> to the BER element <i>ber</i> .		
The ber_put_boolean() function writes the boolean value given by <i>bool</i> to the BER element.		
The ber_put_bitstring() function writes <i>blen</i> bits starting at <i>str</i> as a bitstring value to the given BER element. Note that <i>blen</i> is the length in <i>bits</i> of the bitstring.		
The ber_put_ostring() function writes <i>len</i> bytes starting at <i>str</i> to the BER element as an octet string.		
The ber_put_string() function writes the null-terminated string (minus the terminating ") to the BER element as an octet string.		
The ber_put_null() function writes a NULL element to the BER element.		

The ber\_start\_seq() function is used to start a sequence in the BER element. The ber\_start\_set() function works similarly. The end of the sequence or set is marked by the nearest matching call to ber\_put\_seq() or ber\_put\_set(), respectively.

The ber\_first\_element() function is used to return the tag and length of the first element in a set or sequence. It also returns in *cookie* a magic cookie parameter that should be passed to subsequent calls to ber\_next\_element(), which returns similar information.

**EXAMPLES EXAMPLE 1** Assuming the following variable declarations, and that the variables have been assigned appropriately, an BER encoding of the following ASN.1 object:

```
AlmostASearchRequest := SEQUENCE {
                                 baseObject DistinguishedName,
                                 scope
                                                  ENUMERATED {
                                    baseObject (0),
                                     singleLevel (1),
                                     wholeSubtree (2)
                                 },
                                 derefAliases ENUMERATED {
                                    neverDerefaliases (0),
                                     derefInSearching (1),
                                     derefFindingBaseObj (2),
                                     alwaysDerefAliases (3N)
                                 },
                                 sizelimit INTEGER (0 .. 65535),
timelimit INTEGER (0 .. 65535),
attrsOnly BOOLEAN,
attributes SEQUENCE OF AttributeType
                             }
                      can be achieved like so:
                                    scope, ali, size, time, attrsonly;
                             int
                             char *dn, **attrs;
                             /* ... fill in values ... */
                            if ( (ber = ber_alloc( )) == NULLBER )
                             /* error */
                            if ( ber_printf( ber, "{siiiib{v}}", dn, scope, ali,
                                 size, time, attrsonly, attrs ) == -1 )
                                     /* error */
                            else
                                     /* success */
RETURN VALUES
                      If an error occurs during encoding, ber alloc() returns NULL; other functions
                      generally return -1.
     ATTRIBUTES
                      See attributes(5) for a description of the following attributes:
```

Networking Library Functions 29

ber\_encode(3LDAP)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWlldap (32-bit)
		SUNWldapx (64-bit)
	Stability Level	Evolving
SEE ALSO	<pre>attributes(5), ber_decode(3LDAP)</pre>	
	Yeong, W., Howes, T., and Hardcastle-Kille, Protocol", OSI-DS-26, April 1992.	, S., "Lightweight Directory Access
	Information Processing - Open Systems Inte Service Definition - Specification of Basic En One, International Organization for Standar	ncoding Rules for Abstract Syntax Notation
NOTES	The return values for all of these functions	are declared in the <lber.h> header file.</lber.h>

NAME	bind – bind a name to a socket		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>int bind(int s, const struct sockaddr *name, int namelen);</pre>		
DESCRIPTION	<pre>bind() assigns a name to an unnamed socket. When a socket is created with socket(3SOCKET), it exists in a name space (address family) but has no name assigned.bind() requests that the name pointed to by <i>name</i> be assigned to the socket.</pre>		
RETURN VALUES	If the bind is successful, 0 is returned. A return value of -1 indicates an error, which is further specified in the global errno.		
ERRORS	The bind() call will fail if:		
	EACCES	The requested address is protected and the current user has inadequate permission to access it.	
	EADDRINUSE	The specified address is already in use.	
	EADDRNOTAVAIL	The specified address is not available on the local machine.	
	EBADF	s 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	s is a descriptor for a file, not a socket.	
	The following errors are specific to binding names in the UNIX domain:EACCESSearch permission is denied for a component of the path prefix of the pathname in <i>name</i> .EIOAn I/O error occurred while making the directory entry or allocating the inode.		
	EISDIR	A null pathname was specified.	
	ELOOP	Too many symbolic links were encountered in translating the pathname in <i>name</i> .	
	ENCENT 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.	

### bind(3SOCKET)

	EROFS The inode v	vould reside on a read-only file system.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	MT-Level	Safe	
SEE ALSO	unlink(2), socket(3SOCKET), attribut	ces(5), socket(3HEAD)	
NOTES	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 betwo	-	

NAME	bind – bind a name to a socket		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	<pre>int bind(int socket, const struct sockaddr *address, socklen_t</pre>		
DESCRIPTION	The bind() function assigns an <i>address</i> to an unnamed socket. Sockets created with socket(3XNET) function are initially unnamed; they are identified only by their address family.		
	The function takes the following arguments:		
	socket	Specifies the file descriptor of the socket to be bound.	
	address	Points to a sockaddr structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.	
	address_len	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.	
	The socket in use may require the process to have appropriate privileges to use the bind() function.		
USAGE	An application program can retrieve the assigned socket name with the getsockname(3XNET) function.		
<b>RETURN VALUES</b>	Upon successful completion, bind() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The bind() function will fail if:		
	EADDRINUSE	The specified address is already in use.	
	EADDRNOTAVAIL	The specified address is not available from the local machine.	
	EAFNOSUPPORT	The specified address is not a valid address for the address family of the specified socket.	
	EBADF	The <i>socket</i> argument is not a valid file descriptor.	
	EFAULT	The <i>address</i> argument can not be accessed.	
	EINVAL	The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.	
	ENOTSOCK	The <i>socket</i> argument does not refer to a socket.	
	EOPNOTSUPP	The socket type of the specified socket does not support binding to an address.	

#### bind(3XNET)

 $\mid$  If the address family of the socket is AF\_UNIX, then bind() will fail if:

	EACCES	permission,	nt of the path prefix denies search or the requested name requires writing in a ith a mode that denies write permission.
	EDESTADDRREQ		
	EISDIR	The address a	argument is a null pointer.
	EIO	An I/O erro	r occurred.
	ELOOP		ymbolic links were encountered in he pathname in <i>address.</i>
	ENAMETOOLONG	-	nt of a pathname exceeded NAME_MAX or an entire pathname exceeded PATH_MAX
	ENOENT		nt of the pathname does not name an or the pathname is an empty string.
	ENOTDIR	-	nt of the path prefix of the pathname in t a directory.
	EROFS	The name w	ould reside on a read-only filesystem.
	The bind() function may fai	l if:	
	EACCES		d address is protected and the current user ve permission to bind to it.
	EINVAL	The <i>address_</i> address fam	<i>len</i> argument is not a valid length for the ily.
	EISCONN	The socket is	s already connected.
	ENAMETOOLONG		esolution of a symbolic link produced an e result whose length exceeds PATH_MAX.
	ENOBUFS	Insufficient i call.	resources were available to complete the
	ENOSR	There were a operation to	insufficient STREAMS resources for the complete.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		following attributes:
	ATTRIBUTE TYPE	6	ATTRIBUTE VALUE
	MT-Level		MT-Safe
	1		

SEE ALSO connect(3XNET), getsockname(3XNET), listen(3XNET), socket(3XNET), attributes(5)

34 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

NAME	byteorder, htonl, htons, ntohl, ntohs – convert values between host and network byte order		
SYNOPSIS	<pre>#include <sys types.h=""> #include <netinet in.h=""> #include <inttypes.h></inttypes.h></netinet></sys></pre>		
	<pre>uint32_t htonl(unint32_t hostlong);</pre>		
	<pre>uint16_t htons (uint16_t hostshort);</pre>		
	<pre>uint32_t ntohl (uint32_t netlong);</pre>		
	<pre>uint16_t ntohs(uint16_t netshort);</pre>		
DESCRIPTION	These routines convert 16 and 32 bit quantities between network byte order and host byte order. On some architectures these routines are defined as NULL macros in the include file <netinet in.h="">. On other architectures, if their host byte order is different from network byte order, these routines are functional.</netinet>		
	These routines are most often used in conjure turned by gethostent() and getservegetserveyname(3SOCKET).		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
		ATTRIBUTE VALUE	
	ATTRIBUTE TYPE MT-Level	ATTRIBUTE VALUE Safe	
SEE ALSO		Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	
SEE ALSO	MT-Level gethostbyname(3NSL), getservbyname	Safe	

connect(3SOCKET)			
NAME	connect – initiate a connection on a socket		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>int connect(int s, const struct sockaddr *name, int namelen);</pre>		
DESCRIPTION	The parameter <i>s</i> is a socket. If it is of type SOCK_DGRAM, connect() specifies the peer with which the socket is to be associated; this address is the address to which datagrams are to be sent if a receiver is not explicitly designated; it is the only address from which datagrams are to be received. If the socket <i>s</i> is of type SOCK_STREAM, connect() attempts to make a connection to another socket. The other socket is specified by <i>name</i> . <i>name</i> is an address in the communication space of the socket. Each communication space interprets the <i>name</i> parameter in its own way. If <i>s</i> is not bound, then it will be bound to an address selected by the underlying transport provider. Generally, stream sockets may successfully connect() only once; datagram sockets may use connect() multiple times to change their association. Datagram sockets may dissolve the association by connecting to a null address.		
RETURN VALUES	If the connection or binding succeeds, 0 is returned. Otherwise, -1 is returned and sets errno to indicate the error.		
ERRORS	The call fails if:		
	EACCES	Search permission is denied for a component of the path prefix of the pathname in <i>name</i> .	
	EADDRINUSE	The address is already in use.	
	EADDRNOTAVAIL	The specified address is not available on the remote machine.	
	EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.	
	EALREADY	The socket is non-blocking and a previous connection attempt has not yet been completed.	
	EBADF	s is not a valid descriptor.	
	ECONNREFUSED	The attempt to connect was forcefully rejected. The calling program should close(2) the socket descriptor, and issue another socket(3SOCKET) call to obtain a new descriptor before attempting another connect() call.	
	EINPROGRESS	The socket is non-blocking and the connection cannot be completed immediately. It is possible to select(3C) for completion by selecting the socket for writing. However, this is only possible if the socket STREAMS	

36 man pages section 3: Networking Library Functions • Last Revised 22 Oct 1999

# connect(3SOCKET)

		module is the topmost module on the protocol stack with a write service procedure. This will be the normal case.
	EINTR	The connection attempt was interrupted before any data arrived by the delivery of a signal.
	EINVAL	<i>namelen</i> is not the size of a valid address for the specified address family.
	EIO	An I/O error occurred while reading from or writing to the file system.
	EISCONN	The socket is already connected.
	ELOOP	Too many symbolic links were encountered in translating the pathname in <i>name</i> .
	ENETUNREACH	The network is not reachable from this host.
	ENOENT	A component of the path prefix of the pathname in <i>name</i> does not exist.
	ENOENT	The socket referred to by the pathname in <i>name</i> does not exist.
	ENOSR	There were insufficient STREAMS resources available to complete the operation.
	ENXIO	The server exited before the connection was complete.
	ETIMEDOUT	Connection establishment timed out without establishing a connection.
	EWOULDBLOCK	The socket is marked as non-blocking, and the requested operation would block.
		fic to connecting names in the UNIX domain. These e versions of the UNIX IPC domain.
	ENOTDIR	A component of the path prefix of the pathname in <i>name</i> is not a directory.
	ENOTSOCK	<i>s</i> is not a socket.
	ENOTSOCK	<i>name</i> is not a socket.
	EPROTOTYPE	The file referred to by <i>name</i> is a socket of a type other than type <i>s</i> (for example, <i>s</i> is a SOCK_DGRAM socket, while <i>name</i> refers to a SOCK_STREAM socket).
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:

# connect(3SOCKET)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

# SEE ALSO close(2), accept(3SOCKET), getsockname(3SOCKET), select(3C), socket(3SOCKET), attributes(5), socket(3HEAD)

NAME	NAME   connect – connect a socket		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	<pre>int connect(int socket, const struct sockaddr *address, socklen_t</pre>		
DESCRIPTION	The connect() for takes the following	unction requests a connection to be made on a socket. The function g arguments:	
	socket	Specifies the file descriptor associated with the socket.	
	address	Points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.	
	address_len	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.	
		ot already been bound to a local address, connect() will bind it to unless the socket's address family is AF_UNIX, is an unused local	
	If the initiating socket is not connection-mode, then connect() sets the socket's peer address, but no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent send(3XNET) calls, and limits the remote sender for subsequent recv(3XNET) calls. If <i>address</i> is a null address for the protocol, the socket's peer address will be reset.		
		ket is connection-mode, then connect () attempts to establish a address specified by the <i>address</i> argument.	
	If the connection cannot be established immediately and O_NONBLOCK is not set for the file descriptor for the socket, connect() will block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, connect() will fail and the connection attempt will be aborted. If connect() is interrupted by a signal that is caught while blocked waiting to establish a connection, connect() will fail and set errno to EINTR, but the connection request will not be aborted, and the connection will be established asynchronously.		
	file descriptor for t but the connection asynchronously. So	annot be established immediately and O_NONBLOCK is set for the he socket, connect() will fail and set errno to EINPROGRESS, request will not be aborted, and the connection will be established ubsequent calls to connect() for the same socket, before the blished, will fail and set errno to EALREADY.	
		on has been established asynchronously, select(3C) and poll(2) he file descriptor for the socket is ready for writing.	
		Networking Library Functions 39	

connect(3XNET)		
	The socket in use may require the process to have appropriate privileges to use the connect() function.	
USAGE	If connect() fails, the state of the socket is unspecified. Portable applications should close the file descriptor and create a new socket before attempting to reconnect.	
<b>RETURN VALUES</b>	Upon successful completion, connect() returns 0. Otherwise, -1 is returned and errno is set to indicate the error. The connect() function will fail if:	
ERRORS		
	EADDRNOTAVAIL	The specified address is not available from the local machine.
	EAFNOSUPPORT	The specified address is not a valid address for the address family of the specified socket.
	EALREADY	A connection request is already in progress for the specified socket.
	EBADFThe socket argument is not a valid file descriptor.ECONNREFUSEDThe target address was not listening for connections refused the connection request.EFAULTThe address parameter can not be accessed.EINPROGRESSO_NONBLOCK is set for the file descriptor for the socket and the connection cannot be immediately established; the connection will be established 	
	EISCONN The specified socket is connection-mode and is alread connected.	
	ENETUNREACH	No route to the network is present.
	ENOTSOCK       The socket argument does not refer to a socket.         EPROTOTYPE       The specified address has a different type than the socket bound to the specified peer address.         ETIMEDOUT       The attempt to connect timed out before a connection was made.	
	If the address family of the socket is AF_UNIX, then connect() will fail if:	
	EIO	An I/O error occurred while reading from or writing to the file system.

	ELOOP	Too many symbolic links were encountered in translating the pathname in <i>address</i> .
	ENAMETOOLONG	A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
	ENOENT	A component of the pathname does not name an existing file or the pathname is an empty string.
	ENOTDIR	A component of the path prefix of the pathname in <i>address</i> is not a directory.
	The connect() function may	y fail if:
	EACCES	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
	EADDRINUSE	Attempt to establish a connection that uses addresses that are already in use.
	ECONNRESET	Remote host reset the connection request.
	EHOSTUNREACH	The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
	EINVAL	The <i>address_len</i> argument is not a valid length for the address family; or invalid address family in sockaddr structure.
	ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
	ENETDOWN	The local interface used to reach the destination is down.
	ENOBUFS	No buffer space is available.
	ENOSR	There were insufficient STREAMS resources available to complete the operation.
	EOPNOTSUPP	The socket is listening and can not be connected.
ES	See attributes(5) for descri	ptions of the following attributes:

 $\ensuremath{\textbf{ATTRIBUTES}}\xspace$   $\ensuremath{|}\xspace$  See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

Networking Library Functions 41

connect(3XNET)
----------------

SEE ALSO close(2), poll(2), accept(3XNET), bind(3XNET), getsockname(3XNET), select(3C), send(3XNET), shutdown(3XNET), socket(3XNET), attributes(5)

 NAME
 dial – establish an outgoing terminal line connection

 SYNOPSIS
 cc [ flag ... ] file ... -lnsl [ library ... ]

 #include <dial.h>
 int dial (CALL call);

 void undial (int fd);
 void undial (int fd);

**DESCRIPTION** dial() returns a file-descriptor for a terminal line open for read/write. The argument to dial() is a CALL structure (defined in the header <dial.h>).

When finished with the terminal line, the calling program must invoke undial() to release the semaphore that has been set during the allocation of the terminal device.

CALL is defined in the header <dial.h> and has the following members:

struct termio int int char	<pre>*attr; baud; speed; *line;</pre>	<pre>/* pointer to termio attribute struct */ /* transmission data rate */ /* 212A modem: low=300, high=1200 */ /* device name for out-going line */</pre>
char int char int	<pre>*telno; modem; *device; dev_len;</pre>	<pre>/* pointer to tel-no digits string */    /* specify modem control for direct lines */    /* unused */    /* unused */</pre>

The CALL element speed is intended only for use with an outgoing dialed call, in which case its value should be the desired transmission baud rate. The CALL element baud is no longer used.

If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the line element in the CALL structure. Legal values for such terminal device names are kept in the Devices file. In this case, the value of the baud element should be set to -1. This value will cause dial to determine the correct value from the <Devices > file.

The telno element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of these characters:

0-9	dial 0-9
*	dail *
#	dail #
=	wait for secondary dial tone
-	delay for approximately 4 seconds

The CALL element modem is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element attr is a pointer to a termio structure, as defined in the header <termio.h>. A NULL value

Networking Library Functions 43

dial(3NSL)			
	for this pointer element may be passed to the dial function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This setting is often important for certain attributes such as parity and baud-rate.		
	The CALL elements device and dev_len a CALL structure for compatibility reasons.	are no longer used. They are retained in the	
RETURN VALUES	On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the header <dial.h>.</dial.h>		
FILES	<pre>INTRPT -1 /* interrupt occurred * D_HUNG -2 /* dialer hung (no retu NO_ANS -3 /* no answer within 10 ILL_BD -4 /* illegal baud-rate */ A_PROB -5 /* acu problem (open()) L_PROB -6 /* line problem (open()) L_PROB -6 /* line problem (open()) NO_Ldv -7 /* can't open Devices f DV_NT_A -8 /* requested device not DV_NT_K -9 /* requested device not NO_BD_A -10 /* no device available NO_BD_K -11 /* no device known at r DV_NT_E -12 /* requested speed does BAD_SYS -13 /* system not in System /etc/uucp/Devices /etc/uucp/Systems /var/spool/uucp/LCKtty-device See attributes(5) for descriptions of the speed device of the speed speed device</pre>	<pre>rm from write) */ seconds */ failure) */ ) failure) */ ille */ available */ known */ at requested baud */ requested baud */ sequested baud */ sequested</pre>	
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level	Unsafe	
SEE ALSO NOTES	<pre>uucp(1C), alarm(2), read(2), write(2), at Including the header <dial.h> automatica alarm(2) system call for 3600 seconds is ma for the purpose of "touching" the LCK. fil semaphore for the terminal device. Otherwis entry on its 90-minute clean-up rounds. The is in a read(2) or write(2) function, causin program expects to be around for an hour of be checked for (errno==EINTR), and the This interface is unsafe in multithreaded ap called only from the main thread.</dial.h></pre>	ally includes the header <termio.h>. An ade (and caught) within the dial module be and constitutes the device allocation ise, uucp(1C) may simply delete the LCK e alarm may go off while the user program be an apparent error return. If the user or more, error returns from read()s should read() possibly reissued.</termio.h>	

**NAME** | doconfig – execute a configuration script **SYNOPSIS cc** [ flag ... ] file ... -lnsl [ library ... ] # include <sac.h> int doconfig(int fildes, char \*script, long rflag); DESCRIPTION doconfig() is a Service Access Facility library function that interprets the configuration scripts contained in the files </etc/saf/pmtag/ config>, </etc/saf/ sysconfig>, and </etc/saf/pmtag/svctag>, where pmtag specifies the tag associated with the port monitor, and *svctag* specifies the service tag associated with a given service. See pmadm(1M) and sacadm(1M). script is the name of the configuration script; *fildes* is a file descriptor that designates the stream to which stream manipulation operations are to be applied; rflag is a bitmask that indicates the mode in which script is to be interpreted. If *rflag* is zero, all commands in the configuration script are eligible to be interpreted. If *rflag* has the NOASSIGN bit set, the assign command is considered illegal and will generate an error return. If *rflag* has the NORUN bit set, the run and runwait commands are considered illegal and will generate error returns. The configuration language in which script is written consists of a sequence of commands, each of which is interpreted separately. The following reserved keywords are defined: assign, push, pop, runwait, and run. The comment character is #; when a # occurs on a line, everything from that point to the end of the line is ignored. Blank lines are not significant. No line in a command script may exceed 1024 characters. assign *variable=value* Used to define environment variables. *variable* is the name of the environment variable and *value* is the value to be assigned to it. The value assigned must be a string constant; no form of parameter substitution is available. *value* may be quoted. The quoting rules are those used by the shell for defining environment variables. assign will fail if space cannot be allocated for the new variable or if any part of the specification is invalid. push module1[, module2, module3, . . .] Used to push STREAMS modules onto the stream designated by fildes. module1 is the name of the first module to be pushed, *module2* is the name of the second module to be pushed, etc. The command will fail if any of the named modules cannot be pushed. If a module cannot be pushed, the subsequent modules on the same command line will be ignored and modules that have already been pushed will be popped. pop [module] Used to pop STREAMS modules off the designated stream. If pop is invoked with no arguments, the top module on the stream is popped. If an argument is given, modules will be popped one at a time until the named module is at the top of the stream. If the named module is not on the designated stream, the stream is left as it

doconfig(3NSL)			
	was and the command fails. If <i>module</i> is the special keyword ALL, then all modules on the stream will be popped. Note that only modules above the topmost driver are affected.		
	<pre>runwait command The runwait command runs a command and waits for it to complete. command is the pathname of the command to be run. The command is run with /usr/bin/sh -c prepended to it; shell scripts may thus be executed from configuration scripts. The runwait command will fail if command cannot be found or cannot be executed, or if command exits with a non-zero status.</pre>		
	run command The run command is identical to runwait except that it does not wait for command to complete. command is the pathname of the command to be run. run will not fail unless it is unable to create a child process to execute the command.		
	Although they are syntactically indistinguishable, some of the commands available to run and runwait are interpreter built-in commands. Interpreter built-ins are used when it is necessary to alter the state of a process within the context of that process. The doconfig() interpreter built-in commands are similar to the shell special commands and, like these, they do not spawn another process for execution. See sh(1). The built-in commands are:		
	cd ulimit umask		
RETURN VALUES	doconfig() returns 0 if the script was interpreted successfully. If a command in the script fails, the interpretation of the script ceases at that point and a positive number is returned; this number indicates which line in the script failed. If a system error occurs, a value of $-1$ is returned. When a script fails, the process whose environment was being established should <i>not</i> be started.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Unsafe	
SEE ALSO	${\tt sh}(1), {\tt pmadm}(1{\tt M}), {\tt sacadm}(1{\tt M}), {\tt attribut}$	ces(5)	
NOTES	This interface is unsafe in multithreaded ap called only from the main thread.	plications. Unsafe interfaces should be	
	1		

NAME	endhostent, gethostbyaddr, gethostbyname, gethostent, sethostent – network host database functions	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <netdb.h> extern int h_errno;</netdb.h></pre>	
	<pre>void endhostentvoid););</pre>	
	<pre>struct hostent *gethostbyaddr(const void *addr, size_t len, int</pre>	
	<pre>struct hostent *gethostbyname(const char *name);</pre>	
	<pre>struct hostent *gethostentvoid););</pre>	
	<pre>void sethostent(int stayopen);</pre>	
DESCRIPTION	The gethostent(), gethostbyaddr(), and gethostbyname() functions each return a pointer to a hostent structure, the members of which contain the fields of an entry in the network host database.	
	The gethostent() function reads the next entry of the database, opening a connection to the database if necessary.	
	The gethostbyaddr() function searches the database and finds an entry which matches the address family specified by the type argument and which matches the address pointed to by the <i>addr</i> argument, opening a connection to the database if necessary. The <i>addr</i> argument is a pointer to the binary-format (that is, not null-terminated) address in network byte order, whose length is specified by the <i>len</i> argument. The datatype of the address depends on the address family. For an address of type AF_INET, this is an in_addr structure, defined in <netinet in.h="">. For an address of type AF_INET6, there is an in6_addr structure defined in <netinet in.h="">.</netinet></netinet>	
	The gethostbyname() function searches the database and finds an entry which matches the host name specified by the <i>name</i> argument, opening a connection to the database if necessary. If <i>name</i> is an alias for a valid host name, the function returns information about the host name to which the alias refers, and <i>name</i> is included in the list of aliases returned.	
	The sethostent() function opens a connection to the network host database, and sets the position of the next entry to the first entry. If the <i>stayopen</i> argument is non-zero, the connection to the host database will not be closed after each call to gethostent() (either directly, or indirectly through one of the other gethost*() functions).	
	The endhostent() function closes the connection to the database.	
USAGE	The gethostent(), gethostbyaddr(), and gethostbyname() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.	

endhostent(3XNET)			
	These functions are generally used with the Internet address family.		
RETURN VALUES	On successful completion, gethostbyaddr(), gethostbyname() and gethostent() return a pointer to a hostent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.		
	On unsuccessful completion, gethostbyaddr() and gethostbyname() functions set $h\_errno$ to indicate the error.		
ERRORS	No errors are defined for endhostent(), gethostent() and sethostent().		
	The gethostbyaddr() and gethostbyname() functions will fail in the following cases, setting $h_{errno}$ to the value shown in the list below. Any changes to errno are unspecified.		
	HOST_NOT_FOUND	No such hos	st is known.
	NO_DATA	address is av	ecognised the request and the name but no vailable. Another type of request to the for the domain might return an answer.
	NO_RECOVERY	An unexpected be recovered	ted server failure occurred which can not l.
	TRY_AGAIN		y and possibly transient error occurred, lure of a server to respond.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYP	Ξ	ATTRIBUTE VALUE
	MT-Level		Unsafe
SEE ALSO	endservent(3XNET), hton	l(3XNET), ine	et_addr(3XNET),attributes(5)

NAME	endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent – network database functions
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <netdb.h></netdb.h></pre>
	<pre>void endnetent(void);struct netent *getnetbyaddr(in_addr_t net, int type);</pre>
	<pre>struct netent *getnetbyname(const char *name);</pre>
	<pre>struct netent *getnetentvoid););</pre>
	<pre>void setnetent(int stayopen);</pre>
DESCRIPTION	The getnetbyaddr(), getnetbyname() and getnetent(), functions each return a pointer to a netent structure, the members of which contain the fields of an entry in the network database.
	The getnetent() function reads the next entry of the database, opening a connection to the database if necessary.
	The getnetbyaddr() function searches the database from the beginning, and finds the first entry for which the address family specified by type matches the n_addrtype member and the network number <i>net</i> matches the n_net member, opening a connection to the database if necessary. The <i>net</i> argument is the network number in host byte order.
	The getnetbyname() function searches the database from the beginning and finds the first entry for which the network name specified by <i>name</i> matches the n_name member, opening a connection to the database if necessary.
	The setnetent() function opens and rewinds the database. If the <i>stayopen</i> argument is non-zero, the connection to the net database will not be closed after each call to getnetent() (either directly, or indirectly through one of the other getnet*() functions).
	The endnetent() function closes the database.
USAGE	The getnetbyaddr(), getnetbyname() and getnetent(), functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.
	These functions are generally used with the Internet address family.
RETURN VALUES	On successful completion, getnetbyaddr(), getnetbyname() and getnetent(), return a pointer to a netent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.
ERRORS	No errors are defined.

#### endnetent(3XNET)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

#### SEE ALSO attributes(5)

NAME	endprotoent, getprotobynumber, getprotobyname, getprotoent, setprotoent – network protocol database functions
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <netdb.h></netdb.h></pre>
	<pre>void endprotoentvoid););</pre>
	<pre>struct protoent *getprotobyname(const char *name);</pre>
	<pre>struct protoent *getprotobynumber(int proto);</pre>
	<pre>struct protoent *getprotoentvoid););</pre>
	<pre>void setprotoent(int stayopen);</pre>
DESCRIPTION	The getprotobyname(), getprotobynumber() and getprotoent(), functions each return a pointer to a protoent structure, the members of which contain the fields of an entry in the network protocol database.
	The getprotoent() function reads the next entry of the database, opening a connection to the database if necessary.
	The getprotobyname() function searches the database from the beginning and finds the first entry for which the protocol name specified by <i>name</i> matches the p_name member, opening a connection to the database if necessary.
	The getprotobynumber() function searches the database from the beginning and finds the first entry for which the protocol number specified by <i>number</i> matches the p_proto member, opening a connection to the database if necessary.
	The setprotoent() function opens a connection to the database, and sets the next entry to the first entry. If the <i>stayopen</i> argument is non-zero, the connection to the network protocol database will not be closed after each call to getprotoent() (either directly, or indirectly through one of the other getproto*() functions).
	The endprotoent() function closes the connection to the database.
USAGE	The getprotobyname(), getprotobynumber() and getprotoent() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.
	These functions are generally used with the Internet address family.
RETURN VALUES	On successful completion, getprotobyname(), getprotobynumber() and getprotoent() functions return a pointer to a protoent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.
ERRORS	No errors are defined.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

# endprotoent(3XNET)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SEE ALSO** attributes(5)

NAME	endservent, getservbyport, getservbyname, getservent, setservent – network services database functions
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <netdb.h></netdb.h></pre>
	<pre>void endserventvoid););</pre>
	<pre>struct servent *getservbyname(const char *name, const char *proto);</pre>
	<pre>struct servent *getservbyport(int port, const char *proto);</pre>
	<pre>struct servent *getserventvoid););</pre>
	<pre>void setservent(int stayopen);</pre>
DESCRIPTION	The getservbyname(), getservbyport() and getservent() functions each return a pointer to a servent structure, the members of which contain the fields of an entry in the network services database.
	The getservent() function reads the next entry of the database, opening a connection to the database if necessary.
	The getservbyname() function searches the database from the beginning and finds the first entry for which the service name specified by <i>name</i> matches the s_name member and the protocol name specified by <i>proto</i> matches the s_proto member, opening a connection to the database if necessary. If <i>proto</i> is a null pointer, any value of the s_proto member will be matched.
	The getservbyport() function searches the database from the beginning and finds the first entry for which the port specified by <i>port</i> matches the s_port member and the protocol name specified by <i>proto</i> matches the s_proto member, opening a connection to the database if necessary. If <i>proto</i> is a null pointer, any value of the s_proto member will be matched. The <i>port</i> argument must be in network byte order.
	The setservent() function opens a connection to the database, and sets the next entry to the first entry. If the <i>stayopen</i> argument is non-zero, the net database will not be closed after each call to the getservent() function (either directly, or indirectly through one of the other getserv*() functions).
	The endservent () function closes the database.
USAGE	The <i>port</i> argument of getservbyport() need not be compatible with the port values of all address families.
	The getservent(), getservbyname() and getservbyport() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.
	These functions are generally used with the Internet address family.

endservent(3XNET)

**RETURN VALUES** On successful completion, getservbyname(), getservbyport() and getservent() return a pointer to a servent structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

**ERRORS** No errors are defined.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SEE ALSO** endhostent(3XNET), endprotoent(3XNET), htonl(3XNET), inet\_addr(3XNET), attributes(5)

	ethers(3SOCKET)
NAME	ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – Ethernet address mapping operations
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""> #include <net if.h=""> #include <netinet in.h=""> #include <netinet if_ether.h=""></netinet></netinet></net></sys></sys></pre>
	<pre>char *ether_ntoa(struct ether_addr *e);</pre>
	<pre>struct ether_addr *ether_aton(char *s);</pre>
	<pre>int ether_ntohost(char *hostname, struct ether_addr *e);</pre>
	<pre>int ether_hostton(char *hostname, struct ether_addr *e);</pre>
	<pre>int ether_line(char *l, struct ether_addr *e, char *hostname);</pre>
DESCRIPTION	These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.
	The function ether_ntoa() converts a 48 bit Ethernet number pointed to by <i>e</i> to its standard ASCII representation; it returns a pointer to the ASCII string. The representation is of the form $x : x : x : x$ where <i>x</i> is a hexadecimal number between 0 and ff. The function ether_aton() converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.
	The function ether_ntohost() maps an Ethernet number (pointed to by <i>e</i> ) to its associated hostname. The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. Inversely, the function ether_hostton() maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by <i>e</i> . The function also returns zero upon success and non-zero upon failure. In order to do the mapping, both these functions may lookup one or more of the following sources: the ethers file, the NIS maps "ethers.byname" and "ethers.byaddr" and the NIS+ table "ethers". The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4) for details).
	The function ether_line() scans a line (pointed to by <i>l</i> ) and sets the hostname and the Ethernet number (pointed to by <i>e</i> ). The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by ethers(4).

#### ethers(3SOCKET)

#### FILES | /etc/ethers

/etc/nsswitch.conf

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

#### **SEE ALSO** ethers(4), nsswitch.conf(4), attributes(5)

**BUGS** Programs that call ether\_hostton() or ether\_ntohost() routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.

fn\_attr\_bind(3XFN)

NAME	fn_attr_bind – bind a reference to a name ar	nd associate attributes with named object	
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>		
	<pre>int fn_attr_bind(FN_ctx_t *ctx, cc const FN_ref_t *ref, const FN_ exclusive, FN_status_t *status);</pre>		
DESCRIPTION	This operation binds the supplied reference <i>ref</i> to the supplied composite name <i>name</i> relative to <i>ctx</i> , and associates the attributes specified in <i>attrs</i> with the named object. The binding is made in the target context, that is, that context named by all but the terminal atomic part of <i>name</i> . The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.		
	The value of <i>exclusive</i> determines what hap name is already bound in the target context bound, the operation fails. If <i>exclusive</i> is 0, t binding, and, if <i>attrs</i> is not NULL, <i>attrs</i> replat the named object. If <i>attrs</i> is NULL and <i>exclus</i> with the named object are left unchanged.	. If <i>exclusive</i> is nonzero and <i>name</i> is already he new binding replaces any existing aces any existing attributes associated with	
<b>RETURN VALUES</b>	<pre>fn_attr_bind() returns 1 upon success, 0 upon failure.</pre>		
ERRORS	<pre>fn_attr_bind() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). Of special relevance for this operation is the following status code:</pre>		
	FN_E_NAME_IN_USE The supplied name is already in use.		
USAGE	The value of <i>ref</i> cannot be NULL. If the intent is to reserve a name using fn_attr_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference. If multiple sources are updating a reference or attributes associated with a named object, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference, or manipulating attributes associated with the named object.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level MT-Safe		
SEE ALSO	FN_composite_name_t(3XFN), FN_ctx_	t(3XFN), FN_ref_t(3XFN),	

FN\_status\_t(3XFN), fn\_ctx\_bind(3XFN), fn\_ctx\_lookup(3XFN),

fn\_ctx\_unbind(3XFN), xfn\_attributes(3XFN), xfn\_status\_codes(3XFN),
attributes(5)

Networking Library Functions 57

# fn\_attr\_create\_subcontext(3XFN)

NAME	fn_attr_create_subcontext – create a subcon with newly created context	text in a context and associate attributes	
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_ref_t *fn_attr_create_subconto FN_composite_name_t *name, co FN_status_t *status);</pre>	— — —	
DESCRIPTION	This operation creates a new XFN context of the same type as the target context, that is, that context named by all but the terminal atomic component of <i>name</i> , and binds it to the supplied composite name. In addition, attributes given in <i>attrs</i> are associated with the newly created context.		
	The target context must already exist. The r target context using the terminal atomic nar reference to the newly created context.		
RETURN VALUES	fn_attr_create_subcontext() return if the operation fails, it returns a NULL poin		
ERRORS	<pre>fn_attr_create_subcontext() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). Of special relevance for this operation is the following status code:</pre>		
	FN_E_NAME_IN_USE The termina context.	l atomic name already exists in the target	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), fn_attr_bind(3XI		

NAME	fn_attr_ext_search, FN_ext_searchlist_t, fn_ext_searchlist_next, fn_ext_searchlist_destroy – search for names in the specified context(s) whose attributes satisfy the filter
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>
	<pre>FN_ext_searchlist_t *fn_attr_ext_search(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_search_control_t *control, const FN_search_filter_t *filter, FN_status_t *status);</pre>
	<pre>FN_composite_name_t *fn_ext_searchlist_next(FN_ext_searchlist_t     *esl, FN_ref_t **returned_ref, FN_attrset_t **returned_attrs,     FN_status_t *status);</pre>
	<pre>void fn_ext_searchlist_destroy(FN_ext_searchlist_t *esl);</pre>
DESCRIPTION	This set of operations is used to list names of objects whose attributes satisfy the filter expression. The references to which these names are bound and specified attributes and their values may also be returned.
	control encapsulates the option settings for the search. These options are:
	• the scope of the search
	<ul> <li>whether XFN links are followed</li> </ul>
	<ul> <li>a limit on the number of names returned</li> </ul>
	<ul> <li>whether references and specific attributes associated with the named objects that satisfy the filter are returned</li> </ul>
	The scope of the search is one of:
	• the object named <i>name</i> relative to the context <i>ctx</i>
	• the context named <i>name</i> relative to the context <i>ctx</i>
	• the context named <i>name</i> relative to the context <i>ctx</i> ,
	and its subcontexts
	or
	<ul> <li>the context named <i>name</i> relative to the context <i>ctx</i>, and a context implementation-defined set of subcontexts</li> </ul>
	If the value of <i>control</i> is 0, default control option settings are used. The default settings are:
	<ul> <li>scope is search named context</li> <li>links are not followed</li> <li>all names of objects that satisfy the filter are returned</li> <li>references and attributes are not returned</li> </ul>
	The FN_search_control_t type is described in FN_search_control_t(3XFN).

	The filter expression <i>filter</i> in fn_attr_ext_search() is evaluated against the attributes of the objects bound in the scope of the search. The filter evaluates to either TRUE or FALSE. The names and, optionally, the references and attributes of objects whose attributes satisfy the filter are enumerated. If the value of <i>filter</i> is 0, all names within the search scope are enumerated. The FN_search_filter_t type is described in FN_search_filter_t(3XFN).
	The call to fn_attr_ext_search() initiates the search process. It returns a handle to an FN_ext_searchlist_t object that is used to enumerate the names of the objects that satisfy the filter.
	The operation fn_ext_searchlist_next() returns the next name in the enumeration identified by <i>esl</i> ; it also updates <i>esl</i> to indicate the state of the enumeration. If the reference to which the name is bound was requested, it is returned in <i>returned_ref</i> . Requested attributes associated with the name are returned in <i>returned_attrs</i> ; each attribute consists of an attribute identifier, syntax, and value(s). Successive calls to fn_ext_searchlist_next() using <i>esl</i> return successive names and, optionally, their references and attributes, in the enumeration; these calls further update the state of the enumeration.
	The names that are returned are composite names, to be resolved relative to the starting context for the search. This starting context is the context named <i>name</i> relative to <i>ctx</i> unless the scope of the search is only the named object. If the scope of the search is only the named object, the terminal atomic name in <i>name</i> is returned.
	<pre>fn_ext_searchlist_destroy() releases resources used during the enumeration. This may be invoked at any time to terminate the enumeration.</pre>
RETURN VALUES	<pre>fn_attr_ext_search() returns a pointer to an FN_ext_searchlist_t object if the search is successfully initiated; it returns a NULL pointer if the search cannot be initiated or if no named object with attributes whose values satisfy the filter expression is found.</pre>
	<pre>fn_ext_searchlist_next() returns a pointer to an FN_composite_name_t object (see FN_composite_name_t(3XFN)) that is the next name in the enumeration; it returns a NULL pointer if no more names can be returned. If returned_attrs is a NULL pointer, no attributes are returned; otherwise, returned_attrs contains the attributes associated with the named object, as specified in the control parameter to fn_attr_ext_search(). If returned_ref is a NULL pointer, no reference is returned; otherwise, if control specified the return of the reference of the named object, that reference is returned in returned_ref.</pre>
	In the case of a failure, these operations return in the <i>status</i> argument a code indicating the nature of the failure.
ERRORS	<pre>If successful, fn_attr_ext_search() returns a pointer to an FN_ext_searchlist_t object and sets status to FN_SUCCESS.</pre>

	()		
fn_attr_ext_search() returns a NUL returned. <i>status</i> is set in the following wa			
FN_SUCCESS	A named object could not be found whose attributes satisfied the filter expression.		
FN_E_NOT_A_CONTEXT	The object named for the start of the search was not a context and the search scope was the given context or the given context and its subcontexts.		
FN_E_SEARCH_INVALID_FILTER	The filter could not be evaluated TRUE or FALSE, or there was some other problem with the filter.		
FN_E_SEARCH_INVALID_OPTION	A supplied search control option could not be supported.		
FN_E_SEARCH_INVALID_OP	An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation.		
FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the attributes specified in the filter.		
FN_E_INVALID_ATTR_VALUE	A value type in the filter did not match the syntax of the attribute against which it was being evaluated.		
Other status codes are possible as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).			
Each successful call to fn_ext_searchlist_next() returns a name and, optionally, its reference in <i>returned_ref</i> and requested attributes in <i>returned_attrs. status</i> is set in the following way:			
FN_SUCCESS	All requested attributes were returned successfully with the name.		
FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the requested attributes.		
FN_E_INVALID_ATTR_IDENTIFIER	A requested attribute identifier was not in a format acceptable to the naming system, or its contents were not valid for the format specified.		
FN_E_NO_SUCH_ATTRIBUTE	The named object did not have one of the requested attributes.		

Networking Library Functions 61

at_search(3.	XFN)	
	FN_E_INSUFFICIENT_RESOURCES	Insufficient resources are available to return all the requested attributes and their values.
	FN_E_ATTR_NO_PERMISSION FN_E_INVALID_ATTR_IDENTIFIER FN_E_NO_SUCH_ATTRIBUTE FN_E_INSUFFICIENT_RESOURCES	These indicate that some of the requested attributes may have been returned in <i>returned_attrs</i> but one or more of them could not be returned. Use fn_attr_get(3XFN) or fn_attr_multi_get(3XFN) to discover why these attributes could not be returned.
	<pre>If fn_ext_searchlist_next() return next name in the enumeration.</pre>	s a name, it can be called again to get the
	<pre>fn_ext_searchlist_next() returns a NULL pointer if no more names can be returned. status is set in the following way:</pre>	
	FN_SUCCESS	The search has completed successfully.
	FN_E_PARTIAL_RESULT	The enumeration is not yet complete but cannot be continued.
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the attributes specified in the filter.
	FN_E_INVALID_ENUM_HANDLE	The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).
	Other status codes are possible as describ xfn_status_codes(3XFN).	ed in $FN_status_t(3XFN)$ and
USAGE	E The search performed by fn_attr_ext_search() is not ordered in any way, including the traversal of subcontexts. The names enumerated using fn_ext_searchlist_next() are not ordered in any way. Furthermore, there is no guarantee that any two series of enumerations with the same arguments to fn_attr_ext_search() will return the names in the same order.	
		tion of <i>name</i> are followed, regardless of the h starts at the final named object or context.
	If <i>control</i> specifies that the search should a during the search are followed and the te	follow links, XFN link names encountered rminal named object is searched. If the

terminal named object is bound to a context and the scope of the search includes subcontexts, that context and its subcontexts are also searched. For example, if *aname* is bound to an XFN link, *lname*, in a context within the scope of the search, and *aname* is returned by fn\_ext\_searchlist\_next(), this means that the object identified by *lname* satisfied the filter expression. *aname* is returned instead of *lname* because *aname* can always be named relative to the starting context for the search.

If *control* specifies that the search should not follow links, the attributes associated with the names of XFN links are searched. For example, if *aname* is bound to an XFN link, *lname*, in a context within the scope of the search, and *aname* is returned by fn\_ext\_searchlist\_next(), this means that the object identified by *aname* satisfied the filter expression.

When following XFN links, fn\_attr\_ext\_search() may search contexts outside of *scope*. In addition, if the link name's terminal atomic name is bound in a context within *scope*, the operation may return the same object more than once.

XFN does not specify how *control* affects the following of native naming system links during the search.

**EXAMPLES EXAMPLE 1** A sample program of displaying how the fn\_attr\_ext\_search() operation may be used.

The following code fragment illustrates how the fn\_attr\_ext\_search() operation may be used. The code consists of three parts: preparing the arguments for the search, performing the search, and cleaning up.

The first part involves getting the name of the context to start the search and constructing the search filter that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine get\_search\_query. See FN\_search\_filter\_t(3XFN) for the description of *sfilter* and the filter creation operation.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and search filter to fn\_attr\_ext\_search(). This particular call to fn\_attr\_ext\_search() uses the default search control options (by passing in 0 as the *control* argument). This means that the search will be performed in the context named by *target\_name* and that no reference or attributes will be returned. In addition, any XFN links encountered will not be followed and all named objects that satisfy the search () returns *esl*, a handle for enumerating the results of the search (), which returns the name of the object. (The arguments *returned\_ref* and *returned\_attrs* to fn\_ext\_search() did not request them to be returned.)

**EXAMPLE 1** A sample program of displaying how the fn\_attr\_ext\_search() operation may be used. (*Continued*)

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to fn\_ext\_searchlist\_destroy() releases resources reserved for this enumeration. The other calls release the context handle, name, filter, and status objects created earlier.

```
/* Declarations */
FN_ctx_t *ctx;
FN ext searchlist t *esl;
FN_composite_name_t *name;
FN status t *status = fn status create();
FN composite name t *target name = get name from user input();
FN search filter t *sfilter = get search query();
/* Get context handle to Initial Context */
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
/* Initiate search */
if ((esl=fn_attr_ext_search(ctx, target_name,
    /* default controls */ 0, sfilter, status)) == 0) {
    /* report 'status', cleanup, and exit */
}
/* Enumerate names requested */
while (name=fn_ext_searchlist_next(esl, 0, 0, status)) {
    /* do something with 'name' */
    fn_composite_destroy(name);
}
/* check 'status' for reason for end of enumeration */
/* Clean up */
fn ext searchlist destroy(esl);
fn search filter destroy(sfilter);
fn ctx handle destroy(ctx);
fn_composite_name_destroy(target_name);
fn status destroy(status);
11
* Procedure for constructing the filter object for search:
*
      "age" attribute is greater than or equal to 17 AND
*
        less than or equal to 25
    AND the "student" attribute is present.
*/
FN_search_filter_t *
get search query()
{
    extern FN attribute t *attr age;
    extern FN attribute t *attr student;
    FN_search_filter_t *sfilter;
    unsigned int filter_status;
    sfilter = fn search filter create(
        &filter_status,
        "(a >= 17) and (a <= 25) and a",
        attr_age, attr_age, attr_student);
    /* error checking on 'filter status' */
    return (sfilter);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_attrset\_t(3XFN), FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_ref\_t(3XFN), FN\_search\_control\_t(3XFN), FN\_search\_filter\_t(3XFN), FN\_status\_t(3XFN), fn\_attr\_get(3XFN), fn\_attr\_multi\_get(3XFN), xfn\_status\_codes(3XFN), attributes(5)

# fn\_attr\_get(3XFN)

NAME	fn_attr_get – return specified attribute associated with name		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_attribute_t *fn_attr_get(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, unsigned int follow_link, FN_status_t *status);</pre>		
DESCRIPTION	This operation returns the identifier, syntax and values of a specified attribute for the object named <i>name</i> relative to <i>ctx</i> . If <i>name</i> is empty, the attribute associated with <i>ctx</i> is returned.		
	The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.		
RETURN VALUES	fn_attr_get returns a pointer to an FN_attribute_t object if the operation succeeds; it returns a NULL pointer (0) if the operation fails.		
ERRORS	<pre>fn_attr_get() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).</pre>		
USAGE	<pre>fn_attr_get_values() and its related operations are used for getting individual values of an attribute. They should be used if the combined size of all the values are expected to be too large to be returned in a single invocation of fn_attr_get().</pre>		
	5	if the combined size of all the values are	
ATTRIBUTES	5	if the combined size of all the values are single invocation of fn_attr_get().	
ATTRIBUTES	expected to be too large to be returned in a	if the combined size of all the values are single invocation of fn_attr_get().	
ATTRIBUTES	expected to be too large to be returned in a See attributes(5) for descriptions of the	<pre>if the combined size of all the values are single invocation of fn_attr_get(). following attributes:</pre>	
ATTRIBUTES SEE ALSO	expected to be too large to be returned in a See attributes(5) for descriptions of the ATTRIBUTE TYPE	<pre>if the combined size of all the values are single invocation of fn_attr_get(). following attributes:  ATTRIBUTE VALUE MT-Safe </pre>	

NAME	fn_attr_get_ids – get a list of the identifiers of all attributes associated with named object		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name, unsigned int follow_link, FN_status_t *status);</pre>		
DESCRIPTION	This operation returns a list of the attribute identifiers of all attributes associated with the object named by <i>name</i> relative to the context <i>ctx</i> . If <i>name</i> is empty, the attribute identifiers associated with <i>ctx</i> are returned.		
	The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.		
RETURN VALUES	This operation returns a pointer to an object of type FN_attrset_t; if the operation fails, a NULL pointer (0) is returned.		
ERRORS	This operation sets <i>status</i> as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).		
USAGE	The attributes in the returned set do not contain the syntax or values of the attributes, only their identifiers.		
UDINGL		ntain the syntax or values of the attributes,	
ATTRIBUTES			
	only their identifiers.		
	only their identifiers. See attributes(5) for descriptions of the	following attributes:	
	only their identifiers. See attributes(5) for descriptions of the           ATTRIBUTE TYPE	following attributes: ATTRIBUTE VALUE MT-Safe ((3XFN), FN_composite_name_t(3XFN), En_attr_get(3XFN), xfn_attributes(3XFN),	

# fn\_attr\_get\_values(3XFN)

NAME	fn_attr_get_values, FN_valuelist_t, fn_valuelist_next, fn_valuelist_destroy – return values of an attribute	
SYNOPSIS	NOPSIS cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn>	
	<pre>FN_valuelist_t *fn_attr_get_values(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, unsigned int follow_link, FN_status_t *status);</pre>	
	<pre>FN_attrvalue_t *fn_valuelist_next(FN_valuelist_t *vl,</pre>	
	<pre>void fn_valuelist_destroy(FN_valuelist_t *vl, FN_status_t *status);</pre>	
DESCRIPTION	This set of operations is used to obtain the values of a single attribute, identified by <i>attribute_id</i> , associated with the object named <i>name</i> , resolved in the context <i>ctx</i> . If <i>name</i> is empty, the attribute values associated with <i>ctx</i> are obtained.	
	The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.	
	The operation fn_attr_get_values() initiates the enumeration process. It returns a handle to an FN_valuelist_t object that can be used to enumerate the values of the specified attribute.	
	The operation fn_valuelist_next() returns a new FN_attrvalue_t object containing the next value in the attribute and may be called multiple times until all values are retrieved. The syntax of the attribute is returned in <i>attr_syntax</i> .	
	The operation fn_valuelist_destroy() is used to release the resources used during the enumeration. This may be invoked before the enumeration has completed to terminate the enumeration.	
	These operations work in a fashion similar to the fn_ctx_list_names() operations.	
RETURN VALUES	<pre>fn_attr_get_values() returns a pointer to an FN_valuelist_t object if the enumeration process is successfully initiated; it returns a NULL pointer if the process failed.</pre>	
	<pre>fn_valuelist_next() returns a NULL pointer if no more attribute values can be returned.</pre>	
	In the case of a failure, these operations set <i>status</i> to indicate the nature of the failure.	
ERRORS	Each successful call to fn_valuelist_next() returns an attribute value. <i>status</i> is set to FN_SUCCESS.	

		m_attr_get_values(3XFN)
	When fn_valuelist_next() returns a values can be returned. <i>status</i> is set in the	
	FN_SUCCESS	The enumeration has completed successfully.
	FN_E_INVALID_ENUM_HANDLE	The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).
	FN_E_PARTIAL_RESULT	The enumeration is not yet complete but cannot be continued.
	In addition to these status codes, other status codes are also possible in calls to these operations. In such cases, <i>status</i> is set as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).	
USAGE	This interface should be used instead of fn_attr_get() if the combined size of all the values is expected to be too large to be returned by fn_attr_get().	
ATTRIBUTES	There may be a relationship between the <i>ctx</i> argument supplied to fn_attr_get_values() and the FN_valuelist_t object it returns. For example, some implementations may store the context handle <i>ctx</i> within the FN_valuelist_t object for subsequent fn_valuelist_next() calls. In general, an fn_ctx_handle_destroy(3XFN) should not be invoked on <i>ctx</i> until the enumeration has terminated. See attributes (5) for descriptions of the following attributes:	
	······································	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	<pre>FN_attribute_t(3XFN), FN_attrvalue_t(3XFN), FN_composite_name_t(3XFN), FN_ctx_t(3XFN), FN_identifier_t(3XFN), FN_status_t(3XFN), fn_attr_get(3XFN), fn_ctx_handle_destroy(3XFN), fn_ctx_list_names(3XFN), xfn(3XFN), xfn_attributes(3XFN), xfn_status_codes(3XFN), attributes(5)</pre>	
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of	

# fn\_attr\_get\_values(3XFN)

Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME	FN_attribute_t, fn_attribute_create, fn_attribute_destroy, fn_attribute_copy, fn_attribute_assign, fn_attribute_identifier, fn_attribute_syntax, fn_attribute_valuecount, fn_attribute_first, fn_attribute_next, fn_attribute_add, fn_attribute_remove – an XFN attribute
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_attribute_t *fn_attribute_create(constFN_identifier_t</pre>
	<pre>void fn_attribute_destroy(FN_attribute_t *attr);</pre>
	<pre>FN_attribute_t *fn_attribute_copy(constFN_attribute_t *attr);</pre>
	<pre>FN_attribute_t *fn_attribute_assign(FN_attribute_t *dst, const FN_attribute_t *src);</pre>
	<pre>const FN_identifier_t  *fn_attribute_identifier(constFN_attribute_t *attr);</pre>
	<pre>const FN_identifier_t *fn_attribute_syntax(constFN_attribute_t      *attr);</pre>
	unsigned int <b>fn_attribute_valuecount</b> (constFN_attribute_t *attr);
	<pre>const FN_attrvalue_t *fn_attribute_first(constFN_attribute_t</pre>
	<pre>const FN_attrvalue_t *fn_attribute_next(constFN_attribute_t *attr,</pre>
	<pre>int fn_attribute_add(FN_attribute_t *attr, const FN_attrvalue_t</pre>
	<pre>int fn_attribute_remove(FN_attribute_t *attr, const FN_attrvalue_t      *attribute_value);</pre>
DESCRIPTION	An attribute has an attribute identifier, a syntax, and a set of distinct values. Each value is a sequence of octets. The operations associated with objects of type FN_attribute_t allow the construction, destruction, and manipulation of an attribute and its value set.
	The attribute identifier and its syntax are specified using an FN_identifier_t. fn_attribute_create() creates a new attribute object with the given identifier and syntax, and an empty set of values. fn_attribute_destroy() releases the storage associated with <i>attr</i> . fn_attribute_copy() returns a copy of the object pointed to by <i>attr</i> . fn_attribute_assign() makes a copy of the attribute object pointed to by <i>src</i> and assigns it to <i>dst</i> , releasing any old contents of <i>dst</i> . A pointer to the same object as <i>dst</i> is returned.

# FN\_attribute\_t(3XFN)

X	<pre>fn_attribute_identifier() returns th fn_attribute_syntax() returns the attri fn_attribute_valuecount() returns th </pre>	ribute syntax of <i>attr</i> .
	<pre>fn_attribute_first() and fn_attribute_next() are used to enumerate the values of an attribute. Enumeration of the values of an attribute may return the values in any order. fn_attribute_first() returns an attribute value from attr and sets the iteration marker iter_pos. Subsequent calls to fn_attribute_next() returns the next attribute value identified by iter_pos and advances iter_pos. Adding or removing values from an attribute invalidates any iteration markers that the caller holds.</pre>	
	fn_attribute_add() adds a new value <i>attribute_value</i> to <i>attr</i> . The operation succeeds (but no change is made) if <i>attribute_value</i> is already in <i>attr</i> and <i>exclusive</i> is 0; the operation fails if <i>attribute_value</i> is already in <i>attr</i> and <i>exclusive</i> is non-zero.	
	<pre>fn_attribute_remove() removes attribute_value from attr. The operation succeeds even if attribute_value is not amongst attr's values.</pre>	
RETURN VALUES	<pre>fn_attribute_first() returns 0 if the attribute contains no values. fn_attribute_next() returns 0 if there are no more values to be returned in the attribute (as identified by the iteration marker) or if the iteration marker is invalid.</pre>	
	<pre>fn_attribute_add() and fn_attribute_remove() return 1 if the operation succeeds, 0 if it fails.</pre>	
USAGE	Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in xfn_attributes(3XFN).	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	<pre>FN_attrset_t(3XFN), FN_attrvalue_t(3XFN), FN_identifier_t(3XFN), fn_attr_get(3XFN), fn_attr_modify(3XFN), xfn(3XFN), xfn_attributes(3XFN), attributes(5)</pre>	
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.	

NAME	fn_attr_modify – modify specified attribu	ate associated with name
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>	
	<pre>int fn_attr_modify(FN_ctx_t *ctx, const FN_composite_name_t *name, unsigned int mod_op, const FN_attribute_t *attr, unsigned int follow_link, FN_status_t *status);</pre>	
DESCRIPTION	This operation modifies according to <i>mod_op</i> the attribute <i>attr</i> associated with the object named <i>name</i> relative to <i>ctx</i> . If <i>name</i> is empty, the attribute associated with <i>ctx</i> is modified.	
	The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.	
	The modification is made on the attribute The syntax and values of <i>attr</i> are used ac	e identified by the attribute identifier of <i>attr</i> . cording to the modification operation.
	The modification operations are as follow	vs:
	FN_ATTR_OP_ADD	Add an attribute with given attribute identifier and set of values. If an attribute with this identifier already exists, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits.
	FN_ATTR_OP_ADD_EXCLUSIVE	Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier already exists. The set of values may be empty if the target naming system permits.
	FN_ATTR_OP_REMOVE	Remove the attribute with the given attribute identifier and all of its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored.
	FN_ATTR_OP_ADD_VALUES	Add the given values to those of the given attribute (resulting in the attribute having the union of its prior value set with the set given). Create the attribute if it does not exist already. The set of values may be empty if the target naming system permits.

	N)	
		Remove the given values from those of the given attribute (resulting in the attribute having the set difference of its prior value set and the set given). This succeeds even if some of the given values are not in the set of values that the attribute has. In naming systems that require an attribute to have at east one value, removing the last value will remove the attribute as well.
<b>RETURN VALUES</b>	1 Successful operation.	
	0 Operation failed.	
ERRORS	<pre>fn_attr_modify() sets status as describe xfn_status_codes(3XFN).</pre>	ed in FN_status_t(3XFN) and
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	<pre>FN_attribute_t(3XFN), FN_composite_name_t(3XFN), FN_ctx_t(3XFN), FN_status_t(3XFN), fn_attr_multi_modify(3XFN), xfn(3XFN), xfn_attributes(3XFN), xfn_status_codes(3XFN), attributes(5)</pre>	
NOTES	<pre>FN_status_t(3XFN), fn_attr_multi_modify(3XFN), xfn(3XFN), xfn_attributes(3XFN), xfn_status_codes(3XFN), attributes(5)</pre>	

74 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

NAME	FN_attrmodlist_t, fn_attrmodlist_create, fn_attrmodlist_destroy, fn_attrmodlist_copy, fn_attrmodlist_assign, fn_attrmodlist_count, fn_attrmodlist_first, fn_attrmodlist_next, fn_attrmodlist_add – a list of attribute modifications
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_attrmodlist_t *fn_attrmodlist_create(void);</pre>
	<pre>void fn_attrmodlist_destroy(FN_attrmodlist_t *modlist);</pre>
	<pre>FN_attrmodlist_t *fn_attrmodlist_copy(const FN_attrmodlist_t</pre>
	<pre>FN_attrmodlist_t *fn_attrmodlist_assign(FN_attrmodlist_t *dst,</pre>
	<pre>unsigned int fn_attrmodlist_count(const FN_attrmodlist_t *modlist);</pre>
	<pre>const FN_attribute_t *fn_attrmodlist_first(const     FN_attrmodlist_t *modlist, void **iter_pos, unsigned int     *first_mod_op);</pre>
	<pre>const FN_attribute_t *fn_attrmodlist_next(const FN_attrmodlist_t  *modlist, void **iter_pos, unsigned int *mod_op);</pre>
	<pre>int fn_attrmodlist_add(FN_attrmodlist_t *modlist, unsigned int</pre>
DESCRIPTION	An attribute modification list allows for multiple modification operations to be made on the attributes associated with a single named object. It is used in the fn_attr_multi_modify(3XFN) operation.
	An attribute modification list is a list of attribute modification specifiers. An attribute modification specifier consists of an attribute object and an operation specifier. The attribute's identifier indicates the attribute that is to be operated upon. The attribute's values are used in a manner depending on the operation. The operation specifier is an unsigned int that must have one of the values:
	FN_ATTR_OP_ADD FN_ATTR_OP_ADD_EXCLUSIVE FN_ATTR_OP_REMOVE FN_ATTR_OP_ADD_VALUES
	or
	FN_ATTR_OP_REMOVE_VALUES
	(See fn_attr_modify(3XFN) for detailed descriptions of these specifiers.) The operations are to be performed in the order in which they appear in the modification list.

## FN\_attrmodlist\_t(3XFN)

	<pre>fn_attrmodlist_create() creates an e fn_attrmodlist_destroy() releases th fn_attrmodlist_copy() returns a copy fn_attrmodlist_assign() makes a co old contents of <i>dst</i>. It returns a pointer to the function of the second s</pre>	ne storage associated with <i>modlist</i> . <i>r</i> of the attribute modification list <i>modlist</i> . py of <i>src</i> and assigns it to <i>dst</i> , releasing any
	<pre>fn_attrmodlist_count() returns the n attribute modification list.</pre>	number attribute modification items in the
	The iterators fn_attrmodlist_first() handle to the attribute part of the modifica through an unsigned int * parameter. f attribute of the first modification item from the modification operation of that item; <i>iter</i>	<pre>tion and return the operation specifier part n_attrmodlist_first() returns the modlist and sets mod_op to be the code of</pre>
	<pre>fn_attrmodlist_next() returns the att modlist after iter_pos and advances iter_pos; modification operation of that item. The or enumeration is the same as the order by wh modification list.</pre>	<pre>mod_op is set to the code of the der of the items returned during an</pre>
	fn_attrmodlist_add() adds a new item consisting of the given modification operation code <i>mod_op</i> and attribute <i>attr</i> to the end of the modification list <i>modlist</i> . <i>attr</i> 's identifier indicates the attribute that is to be operated upon. <i>attr</i> 's values are used in a manner depending on the operation.	
RETURN VALUES	<pre>fn_attrmodlist_first() returns 0 if the modification list is empty. fn_attrmodlist_next() returns 0 if there are no more items on the modification list to be enumerated or if the iteration marker is invalid.</pre>	
	fn_attrmodlist_add() returns 1 if the operation succeeds, 0 if the operation fails.	
USAGE	Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in xfn_attributes(3XFN).	
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO FN_attribute_t(3XFN), FN_attrset_t(3XFN), FN_identifier_t(3XFN), fn_attr_modify(3XFN), fn_attr_multi_modify(3XFN), xfn(3XFN), xfn_attributes(3XFN), attributes(5)		<pre>ci_modify(3XFN), xfn(3XFN),</pre>

76 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

FN\_attrmodlist\_t(3XFN)

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

# fn\_attr\_multi\_get(3XFN)

NAME	fn_attr_multi_get, FN_multigetlist_t, fn_multigetlist_next, fn_multigetlist_destroy – return multiple attributes associated with named object
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_multigetlist_t *fn_attr_multi_get(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_attrset_t *attr_ids, unsigned int follow_link, FN_status_t *status);</pre>
	<pre>FN_attribute_t *fn_multigetlist_next(FN_multigetlist_t *ml,</pre>
	<pre>void fn_multigetlist_destroy(FN_multigetlist_t *ml, FN_status_t</pre>
DESCRIPTION	This set of operations returns one or more attributes associated with the object named by <i>name</i> relative to the context <i>ctx</i> . If <i>name</i> is empty, the attributes associated with <i>ctx</i> are returned.
	The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.
	The attributes returned are those specified in <i>attr_ids</i> . If the value of <i>attr_ids</i> is 0, all attributes associated with the named object are returned. Any attribute values in <i>attr_ids</i> provided by the caller are ignored; only the attribute identifiers are relevant for this operation. Each attribute (identifier, syntax, values) is returned one at a time using an enumeration scheme similar to that for listing a context.
	<pre>fn_attr_multi_get() initiates the enumeration process. It returns a handle to an FN_multigetlist_t object that can be used for the enumeration.</pre>
	The operation fn_multigetlist_next() returns a new FN_attribute_t object containing the next attribute (identifiers, syntaxes, and values) requested and updates <i>ml</i> to indicate the state of the enumeration.
	The operation fn_multigetlist_destroy() releases the resources used during the enumeration. It may be invoked before the enumeration has completed to terminate the enumeration.
RETURN VALUES	<pre>fn_attr_multi_get() returns a pointer to an FN_multigetlist_t object if the enumeration has been initiated successfully; a NULL pointer (0) is returned if it failed.</pre>
	<pre>fn_multigetlist_next() returns a pointer to an FN_attribute_t object if an attribute was returned, a NULL pointer (0) if no attribute was returned.</pre>
	In the case of a failure, these operations set <i>status</i> to indicate the nature of the failure.

**ERRORS** | Each call to fn\_multigetlist\_next() sets status as follows:

ERRORS	Lacit call to III_multigetIISt_liext()	sets status as follows.
	FN_SUCCESS	If an attribute was returned, there are more attributes to be enumerated. If no attribute was returned, the enumeration has completed successfully.
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read this attribute.
	FN_E_INSUFFICIENT_RESOURCES	Insufficient resources are available to return the attribute's values.
	FN_E_INVALID_ATTR_IDENTIFIER	This attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier.
	FN_E_INVALID_ENUM_HANDLE	(No attribute should be returned with this status code). The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the object being processed no longer accepts the handle (due to such events as handle expiration or updates to the object's attribute set).
	FN_E_NO_SUCH_ATTRIBUTE	The object did not have an attribute with the given identifier.
	FN_E_PARTIAL_RESULT	(No attribute should be returned with this status code). The enumeration is not yet complete but cannot be continued.
	attribute contains only the attribute ident status codes and FN_SUCCESS (when an	FN_E_NO_SUCH_ATTRIBUTE, the returned ifier (no value or syntax). For these four attribute was returned), ed again to return another attribute. All other
	Other status codes, such as FN_E_COMMU which case, no attribute is returned. In su FN_status_t(3XFN) and xfn_status_	
USAGE	may choose to return only the attributes f FN_E_PARTIAL_RESULT; such implement	rn all attributes requested by <i>attr_ids</i> . Some found successfully, followed by a status of entations may not necessarily return t be read. Implementations are not required

fn\_attr\_multi\_get(3XFN)

There may be a relationship between the *ctx* argument supplied to fn\_attr\_multi\_get() and the FN\_multigetlist\_t object it returns. For example, some implementations may store the context handle *ctx* within the FN\_multigetlist\_t object for subsequent fn\_multigetlist\_next() calls. In general, a fn\_ctx\_handle\_destroy() should not be invoked on *ctx* until the enumeration has terminated.

```
EXAMPLES | EXAMPLE 1 A sample program displaying how to use fn_attr_multi_get() function.
```

The following code fragment illustrates to obtain all attributes associated with a given name using the fn attr multi get() operations.

```
/* list all attributes associated with given name */
                extern FN_string_t *input_string;
                FN ctx t *ctx;
                FN_composite_name_t *target_name = fn_composite_name_from_string(input_string);
                FN multigetlist t *ml;
                FN status t *status = fn status create();
                FN_attribute_t *attr;
                int done = 0;
                ctx = fn ctx handle from initial(status);
                /* error checking on 'status' */
                /* attr_ids == 0 indicates all attributes are to be returned */
                if ((ml=fn_attr_multi_get(ctx, target_name, 0, status)) == 0) {
                    /* report 'status' and exit */
                while ((attr=fn multigetlist next(ml, status)) && !done) {
                    switch (fn status code(status)) {
                    case FN_SUCCESS:
                       /* do something with 'attr' */
                        break;
                    case FN E ATTR NO PERMISSION:
                    case FN_E_ATTR_INVALID_ATTR_IDENTIFIER:
                    case FN E NO SUCH ATTRIBUTE:
                        /* report error using identifier in 'attr' */
                        break;
                    default:
                        /* other error handling */
                        done = 1;
                    if (attr)
                        fn attribute destroy(attr);
                /* check 'status' for reason for end of enumeration and report if necessary */
                /* clean up */
                fn multigetlist destroy(ml, status);
                /* report 'status' */
ATTRIBUTES
                See attributes(5) for descriptions of the following attributes:
```

```
80 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996
```

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	FN_attribute_t(3XFN), FN_attrset_t FN_ctx_t(3XFN), FN_identifier_t(3XFn_attr_get(3XFN), fn_ctx_handle_d fn_ctx_list_names(3XFN), xfn(3XFN), xfn_status_codes(3XFN), attributes	FN), FN_status_t(3XFN), lestroy(3XFN), xfn_attributes(3XFN),

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

### fn\_attr\_multi\_modify(3XFN)

NAME	fn_attr_multi_modify – modify multiple at	tributes associated with named object
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>	
	<pre>int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t     *name, const FN_attrmodlist_t *mods, unsigned int follow_link,     FN_attrmodlist_t **unexecuted_mods, FN_status_t *status);</pre>	
DESCRIPTION	This operation modifies the attributes associated with the object named <i>name</i> relative to <i>ctx</i> . If <i>name</i> is empty, the attributes associated with <i>ctx</i> are modified.	
The value of <i>follow_link</i> determines what happens when the t <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such the values of the attribute associated with the final named ob <i>follow_link</i> is zero, such a link is not followed. Any XFN links terminal		is non-zero, such a link is followed, and the final named object are returned; if
	In the <i>mods</i> parameter, the caller specifies a done in order on the attributes. Each modified modification operation code (see fn_attributes.	
	The FN_attrmodlist_t type is described	d in FN_attrmodlist_t( $3XFN$ ).
RETURN VALUES	fn_attr_multi_modify() returns 1 if all the modification operations were performed successfully. The function returns 0 if it any error occurs. If the operation fails, <i>status</i> and <i>unexecuted_mods</i> are set as described below.	
ERRORS	If an error is encountered while performing the list of modifications, <i>status</i> indicates the type of error and <i>unexecuted_mods</i> is set to a list of unexecuted modifications. The contents of <i>unexecuted_mods</i> do not share any state with <i>mods</i> ; items in <i>unexecuted_mods</i> are copies of items in <i>mods</i> and appear in the same order in which they were originally supplied in <i>mods</i> . The first operation in <i>unexecuted_mods</i> is the first one that failed and the code in <i>status</i> applies to this modification operation in particular. If <i>status</i> indicates failure and a NULL pointer (0) is returned in <i>unexecuted_mods</i> , that indicates no modifications were executed.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	FN_attrmodlist_t(3XFN), FN_composition	ite_name_t(3XFN), FN_ctx_t(3XFN),

FN\_status\_t(3XFN), fn\_attr\_modify(3XFN), xfn(3XFN),

xfn\_attributes(3XFN), xfn\_status\_codes(3XFN), attributes(5)

82 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

## fn\_attr\_search(3XFN)

NAME	fn_attr_search, FN_searchlist_t, fn_searchlist_next, fn_searchlist_destroy – search for the atomic name of objects with the specified attributes in a single context	
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>	
	<pre>FN_searchlist_t *fn_attr_search(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_attrset_t *match_attrs, unsigned int return_ref, const FN_attrset_t *return_attr_ids, FN_status_t *status);</pre>	
	<pre>FN_string_t *fn_searchlist_next(FN_searchlist_t *sl, FN_ref_t</pre>	
	<pre>void fn_searchlist_destroy(FN_searchlist_t *sl);</pre>	
DESCRIPTION	This set of operations is used to enumerate names of objects bound in the target context named <i>name</i> relative to the context <i>ctx</i> with attributes whose values match all those specified by <i>match_attrs</i> .	
	The attributes specified by <i>match_attrs</i> form a conjunctive AND expression against which the attributes of each named object in the target context are evaluated. For multi-valued attributes, the list order of values is ignored and attribute values not specified in <i>match_attrs</i> are ignored. If no value is specified for an attribute in <i>match_attrs</i> , the presence of the attribute is tested. If the value of <i>match_attrs</i> is 0, all names in the target context are enumerated.	
	If a non-zero value of <i>return_ref</i> is passed to fn_attr_search(), the reference bound to the name is returned in the <i>returned_ref</i> argument to fn_searchlist_next().	
	Attribute identifiers and values associated with named objects that satisfy <i>match_attrs</i> may be returned by fn_searchlist_next(). The attributes returned are those listed in the <i>return_attr_ids</i> argument to fn_attr_search(). If the value of <i>return_attr_ids</i> is 0, all attributes are returned. If <i>return_attr_ids</i> is an empty FN_attrset_t(3XFN) object, no attributes are returned. Any attribute values in <i>return_attr_ids</i> are ignored; only the attribute identifiers are relevant for <i>return_attr_ids</i> .	
	The call to fn_attr_search() initiates the enumeration process. It returns a handle to an FN_searchlist_t object that is used to enumerate the names of the objects whose attributes match the attributes specified by <i>match_attrs</i> .	
	The operation fn_searchlist_next() returns the next name in the enumeration identified by the <i>sl</i> . The reference of the name is returned in <i>returned_ref</i> if <i>return_ref</i> was set in the call to fn_attr_search(). The attributes specified by <i>return_attr_ids</i> are returned in <i>returned_attrs</i> . fn_searchlist_next() also updates <i>sl</i> to indicate the state of the enumeration. Successive calls to fn_searchlist_next() using <i>sl</i> return successive names, and optionally, references and attributes, in the enumeration; these calls further update the state of the enumeration.	

84 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

		m_attr_search(0ArN)
	<pre>fn_searchlist_destroy() releases n can be invoked at any time to terminate</pre>	resources used during the enumeration. This the enumeration.
	<pre>fn_attr_search() does not follow XI</pre>	FN links that are bound in the target context.
<b>RETURN VALUES</b> fn_attr_search() returns a pointer to an FN_searchlist_t object enumeration is successfully initiated; it returns a NULL pointer if the enu cannot be initiated or if no named object with attributes whose values m specified in <i>match_attrs</i> is found.		eturns a NULL pointer if the enumeration
	returns a NULL pointer if no more names <i>returned_ref</i> is a NULL pointer, or if the <i>ret</i> reference is returned; otherwise, <i>returned</i>	<i>turn_ref</i> parameter to <i>fn_attr_search</i> was 0, no <i>_ref</i> contains the reference bound to the name. ributes are returned; otherwise, <i>returned_attrs</i> he named object, as specified by the
	In the case of a failure, these operations near the nature of the failure.	return in the <i>status</i> argument a code indicating
ERRORS	fn_attr_search() returns a NULL point of the status argument is set in the following the status argument is set in the status argument in the status argument is set in the status argument is s	inter if the enumeration could not be initiated. ng way:
	FN_SUCCESS	A named object could not be found whose attributes satisfied the implied filter of equality and conjunction.
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the specified attributes.
	FN_E_INVALID_ATTR_VALUE	A value type in the specified attributes did not match the syntax of the attribute against which it was being evaluated.
	Other status codes are possible as describ xfn_status_codes(3XFN).	bed in $FN_status_t(3XFN)$ and
	Each successful call to fn_searchlist_reference and requested attributes. <i>status</i>	_next() returns a name and, optionally, the is set in the following way:
	FN_SUCCESS	All requested attributes were returned successfully with the name.
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the requested attributes.
	FN_E_INVALID_ATTR_IDENTIFIER	A requested attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified.

Networking Library Functions 85

## fn\_attr\_search(3XFN)

	FN_E_NO_SUCH_ATTRIBUTE	The named object did not have one of the requested attributes.
	FN_E_INSUFFICIENT_RESOURCES	Insufficient resources are available to return all the requested attributes and their values.
	FN_E_ATTR_NO_PERMISSION FN_E_INVALID_ATTR_IDENTIFIER FN E NO SUCH ATTRIBUTE	
	FN_E_INSUFFICIENT_RESOURCES	These indicate that some of the requested attributes may have been returned in <i>returned_attrs</i> but one or more of them could not be returned. Use fn_attr_get(3XFN) or fn_attr_multi_get(3XFN) to discover why these attributes could not be returned.
	fn_searchlist_next() returns a NUL The status argument is set in the followir	L pointer if no more names can be returned. ng way:
	FN_SUCCESS	The search has completed successfully.
	FN_E_PARTIAL_RESULT	The enumeration is not yet complete but cannot be continued.
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to read one or more of the specified attributes.
	FN_E_INVALID_ENUM_HANDLE	The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).
	Other status codes are possible as describ xfn_status_codes(3XFN).	ped in $FN_status_t(3XFN)$ and
USAGE		hlist_next() are not ordered in any way. ny two series of enumerations on the same urn the names in the same order.
EXAMPLES	<b>EXAMPLE 1</b> A sample program of displaying	how to use fn_attr_search() function.
	The following code fragment illustrates h be used. The code consists of three parts: performing the search, and cleaning up.	now the fn_attr_search() operation may preparing the arguments for the search,

86 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

**EXAMPLE 1** A sample program of displaying how to use fn\_attr\_search() function. (*Continued*)

The first part involves getting the name of the context to start the search and constructing the set of attributes that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine get\_search\_query.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and matching attributes to fn\_attr\_search(). This particular call to fn\_attr\_search() is requesting that no reference be returned (by passing in 0 for *return\_ref*), and that all attributes associated with the named object be returned (by passing in 0 as the *return\_attr\_ids* argument). If successful, fn\_attr\_search() returns *sl*, a handle for enumerating the results of the search. The results of the search are enumerated using calls to fn\_searchlist\_next(), which returns the name of the object and the attributes associated with the named object in *returned\_attrs*.

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to fn\_searchlist\_destroy() releases resources reserved for this enumeration. The other calls release the context handle, name, attribute set, and status objects created earlier.

```
/* Declarations */
FN_ctx_t *ctx;
FN_searchlist_t *sl;
FN string t *name;
FN_attrset_t *returned_attrs;
FN status t *status = fn status create();
FN composite name t *target name = get name from user input();
FN_attrset_t *match_attrs = get_search_query();
/* Get context handle to Initial Context */
ctx = fn ctx handle from initial(status);
/* error checking on 'status' */
/* Initiate search */
if ((sl=fn_attr_search(ctx, target_name, match_attrs,
   /* no reference */ 0, /* return all attrs */ 0, status)) == 0) {
    /* report 'status', cleanup, and exit */
}
/* Enumerate names and attributes requested */
while (name=fn searchlist next(sl, 0, &returned attrs, status)) {
    /* do something with 'name' and 'returned attrs'*/
    fn_string_destroy(name);
    fn attrset destroy(returned attrs);
/* check 'status' for reason for end of enumeration */
/* Clean up */
fn_searchlist_destroy(sl); /* Free resources of 'sl' */
fn status destroy(status);
fn_attrset_destroy(match_attrs);
fn_ctx_handle_destroy(ctx);
fn composite name destroy(target name);
/*
```

fn\_attr\_search(3XFN)

**EXAMPLE 1** A sample program of displaying how to use fn attr search() function. (Continued) \* Procedure for constructing attribute set containing \* attributes to be matched: \* "zip code" attribute value is "02158" AND "employed" attribute is present. \* \*/ FN attrset t \* get\_search\_query() { /\* Zip code and employed attribute identifier, syntax \*/ extern FN\_attribute\_t \*attr\_zip\_code; extern FN\_attribute\_t \*attr\_employed; FN\_attribute\_t \*zip\_code = fn\_attribute\_copy(attr\_zip\_code); FN\_attr\_value\_t zc\_value = {5, "02158"}; FN\_attrset\_t \*match\_attrs = fn\_attrset\_create(); fn\_attribute\_add(zip\_code, &zc\_value, 0); fn attrset add(match attrs, zip code, 0); fn\_attrset\_add(match\_attrs, attr\_employed, 0); return (match attrs); } **ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_attribute\_t(3XFN), FN\_attrset\_t(3XFN), FN\_attrvalue\_t(3XFN), FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_status\_t(3XFN), FN\_string\_t(3XFN), fn\_attr\_ext\_search(3XFN), fn\_attr\_get(3XFN), fn\_attr\_multi\_get(3XFN), fn\_ctx\_list\_names(3XFN), xfn status codes(3XFN), attributes(5)

88 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

NAME	FN_attrset_t, fn_attrset_create, fn_attrset_destroy, fn_attrset_copy, fn_attrset_assign, fn_attrset_get, fn_attrset_count, fn_attrset_first, fn_attrset_next, fn_attrset_add, fn_attrset_remove – a set of XFN attributes	
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>	
	<pre>FN_attrset_t *fn_attrset_create(void);</pre>	
	<pre>void fn_attrset_destroy(FN_attrset_t *aset);</pre>	
	<pre>FN_attrset_t *fn_attrset_copy(constFN_attrset_t *aset);</pre>	
	<pre>FN_attrset_t *fn_attrset_assign(FN_attrset_t *dst, const FN_attrset_t *src);</pre>	
	<pre>const FN_attribute_t *fn_attrset_get(constconst FN_attrset_t     *aset, const FN_identifier_t *attr_id);</pre>	
	<pre>unsigned int fn_attrset_count(constFN_attrset_t *aset);</pre>	
	<pre>const FN_attribute_t *fn_attrset_first(constFN_attrset_t *aset, void **iter_pos);</pre>	
	<pre>const FN_attribute_t *fn_attrset_next(constFN_attrset_t *aset, void **iter_pos);</pre>	
	<pre>int fn_attrset_add(FN_attrset_t *aset, const FN_attribute_t *attr,</pre>	
	<pre>int fn_attrset_remove(FN_attrset_t *aset, const FN_identifier_t      *attr_id);</pre>	
DESCRIPTION	An attribute set is a set of attribute objects with distinct identifiers. The fn_attr_multi_get(3XFN) operation takes an attribute set as parameter and returns an attribute set. The fn_attr_get_ids(3XFN) operation returns an attribute set containing the identifiers of the attributes.	
	Attribute sets are represented by the type FN_attrset_t. The following operations are defined for manipulating attribute sets.	
	<pre>fn_attrset_create() creates an empty attribute set. fn_attrset_destroy() releases the storage associated with the attribute set aset. fn_attrset_copy() returns a copy of the attribute set aset. fn_attrset_assign() makes a copy of the attribute set src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.</pre>	
	<pre>fn_attrset_get() returns the attribute with the given identifier attr_id from aset. fn_attrset_count() returns the number attributes found in the attribute set aset.</pre>	
	<pre>fn_attrset_first() and fn_attrset_next() are functions that can be used to return an enumeration of all the attributes in an attribute set. The attributes are not ordered in any way. There is no guaranteed relation between the order in which items</pre>	

FN_attrset_	t(3XFN)
-------------	---------

are added to an attribute set and the order of the enumeration. The specification does guarantee that any two enumerations will return the members in the same order, provided that no fn\_attrset\_add() or fn\_attrset\_remove() operation was performed on the object in between or during the two enumerations. fn\_attrset\_first() returns the first attribute from the set and sets *iter\_pos* after the first attribute. fn\_attrset\_next() returns the attribute following *iter\_pos* and advances *iter\_pos*.

fn\_attrset\_add() adds the attribute *attr* to the attribute set aset, replacing the attribute's values if the identifier of *attr* is not distinct in aset and *exclusive* is 0. If *exclusive* is non-zero and the identifier of *attr* is not distinct in aset, the operation fails.

fn\_attrset\_remove() removes the attribute with the identifier attr\_id from aset.
The operation succeeds even if no such attribute occurs in aset.

**RETURN VALUES** fn\_attrset\_first() returns 0 if the attribute set is empty. fn\_attrset\_next() returns 0 if there are no more attributes in the set.

fn\_attrset\_add() and fn\_attrset\_remove() return 1 if the operation
succeeds, and 0 if the operation fails.

- **USAGE** Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in xfn attributes(3XFN).
- **ATTRIBUTES** See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

- SEE ALSO FN\_attribute\_t(3XFN), FN\_attrvalue\_t(3XFN), FN\_identifier\_t(3XFN), fn\_attr\_get\_ids(3XFN), fn\_attr\_multi\_get(3XFN), xfn(3XFN), xfn attributes(3XFN), attributes(5)
  - **NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.
- 90 man pages section 3: Networking Library Functions Last Revised 13 Dec 1996

<pre>NAME FN_attrvalue_t - an XFN attribute value SYNOPSIS cc [fkg ] fiklxfn [ library ] finclude exfn/xfn.h&gt; DESCRIPTION The type FN_attrvalue_t is used to represent the contents of a single attribut value, within an attribute of type FN_attribute_t. The representation of this structure is defined by XFN as follows: typedef struct { size_t length; void 'contents; } FN_attrvalue_t; SEE ALSO FN_attribute_t(3XFN), fn_attr_get_values(3XFN), xfn(3XFN)</pre>		
<pre>#include <xfn xfn.h=""> DESCRIPTION The type FN_attrvalue_t is used to represent the contents of a single attribut value, within an attribute of type FN_attribute_t. The representation of this structure is defined by XFN as follows: typedef struct { size_t length; void *contents; } FN_attrvalue_t;</xfn></pre>	NAME	FN_attrvalue_t – an XFN attribute value
<pre>value, within an attribute of type FN_attribute_t. The representation of this structure is defined by XFN as follows: typedef struct { size_t length; void *contents; } FN_attrvalue_t;</pre>	SYNOPSIS	
<pre>typedef struct { size_t length; void *contents; } FN_attrvalue_t;</pre>	DESCRIPTION	The type FN_attrvalue_t is used to represent the contents of a single attribute value, within an attribute of type FN_attribute_t.
<pre>void *contents; } FN_attrvalue_t;</pre>		The representation of this structure is defined by XFN as follows:
SEE ALSO FN_attribute_t(3XFN), fn_attr_get_values(3XFN), xfn(3XFN)		
	SEE ALSO	<pre>FN_attribute_t(3XFN), fn_attr_get_values(3XFN), xfn(3XFN)</pre>
	I	

## FN\_composite\_name\_t(3XFN)

NAME	FN_composite_name_t, fn_composite_name_create, fn_composite_name_destroy, fn_composite_name_from_str, fn_composite_name_from_string, fn_string_from_composite_name, fn_composite_name_copy, fn_composite_name_assign, fn_composite_name_is_empty, fn_composite_name_count, fn_composite_name_first, fn_composite_name_next, fn_composite_name_prev, fn_composite_name_last, fn_composite_name_prefix, fn_composite_name_suffix, fn_composite_name_is_equal, fn_composite_name_is_prefix, fn_composite_name_is_suffix, fn_composite_name_prepend_comp, fn_composite_name_append_comp, fn_composite_name_insert_comp, fn_composite_name_delete_comp,
	fn_composite_name_prepend_name, fn_composite_name_append_name, fn_composite_name_insert_name – a sequence of component names spanning multiple naming systems
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_composite_name_t *fn_composite_name_create(void);</pre>
	<pre>void fn_composite_name_destroy(FN_composite_name_t *name);</pre>
	<pre>FN_composite_name_t *fn_composite_name_from_str(const unsigned</pre>
	<pre>FN_composite_name_t *fn_composite_name_from_string(const     FN_string_t *str);</pre>
	<pre>FN_string_t *fn_string_from_composite_name(const FN_composite_name_t *name, unsigned int *status);</pre>
	<pre>FN_composite_name_t *fn_composite_name_copy(const     FN_composite_name_t *name);</pre>
	<pre>FN_composite_name_t  *fn_composite_name_assign(FN_composite_name_t *dst, const  FN_composite_name_t *src);</pre>
	<pre>int fn_composite_name_is_empty(const FN_composite_name_t *name);</pre>
	<pre>unsigned int fn_composite_name_count(const FN_composite_name_t     *name);</pre>
	<pre>const FN_string_t *fn_composite_name_first(const FN_composite_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_composite_name_next(const FN_composite_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_composite_name_prev(const     FN_composite_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_composite_name_last(const FN_composite_name_t *name, void **iter_pos);</pre>

92 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

- FN\_composite\_name\_t \*fn\_composite\_name\_prefix(const FN composite name t \*name, const void \*iter\_pos);
- FN\_composite\_name\_t \*fn\_composite\_name\_suffix(const FN composite name t \*name, const void \*iter\_pos);

**DESCRIPTION** A composite name is represented by an object of type FN\_composite\_name\_t. Each component is a string name, of type FN\_string\_t, from the namespace of a single naming system. It may be an atomic name or a compound name in that namespace.

fn\_composite\_name\_create creates an FN\_composite\_name\_t object with zero
components. Components may be subsequently added to the composite name using
the modify operations described below. fn\_composite\_name\_destroy releases any
storage associated with the given FN\_composite\_name\_t handle.

fn\_composite\_name\_from\_str() creates an FN\_composite\_name\_t from the
given null-terminated string based on the code set of the current locale setting, using
the XFN composite name syntax. fn\_composite\_name\_from\_string() creates an
FN\_composite\_name\_t from the string str using the XFN composite name syntax.
fn\_string\_from\_composite\_name() returns the standard string form of the

#### FN\_composite\_name\_t(3XFN)

given composite name, by concatenating the components of the composite name in a left to right order, each separated by the XFN component separator.

fn\_composite\_name\_copy() returns a copy of the given composite name object.
fn\_composite\_name\_assign() makes a copy of the composite name object
pointed to by *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the
same object as *dst* is returned.

fn\_composite\_name\_is\_empty() returns 1 if the given composite name is an
empty composite name (that is, it consists of a single, empty component name);
otherwise, it returns 0. fn\_composite\_name\_count() returns the number of
components in the given composite name.

The iteration scheme is based on the exchange of an opaque void \* argument, *iter\_pos*, that serves to record the position of the iteration in the sequence. Conceptually, *iter\_pos* records a position between two successive components (or at one of the extreme ends of the sequence).

The function fn\_composite\_name\_first() returns a handle to the FN\_string\_t that is the first component in the name, and sets *iter\_pos* to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the fn\_composite\_name\_next() function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, fn\_composite\_name\_next() returns 0. Similarly, fn\_composite\_name\_prev() returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, fn\_composite\_name\_prev() returns 0. The function fn\_composite\_name\_last() returns 0. The function sequence, fn\_composite\_name\_last() returns 0. The function fn\_composite\_name\_last() returns 0. The function fn\_composite\_name\_last() returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to fn\_composite\_name\_prev() can be used to step through leading components of the name).

The fn\_composite\_name\_suffix() function returns a composite name consisting of a copy of those components following the supplied iteration marker. The method fn\_composite\_name\_prefix() returns a composite name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized using fn\_composite\_name\_first(), fn\_composite\_name\_last(), fn\_composite\_name\_is\_prefix(), or fn\_composite\_name\_is\_suffix() yields undefined and generally undesirable behavior.

The functions fn\_composite\_name\_is\_equal(),

fn\_composite\_name\_is\_prefix(), and fn\_composite\_name\_is\_suffix() test for equality between composite names or between parts of composite names. For these functions, equality is defined as exact string equality, not name equivalence. A name's syntactic property, such as case-insensitivity, is not taken into account by these functions.

	The function fn_composite_name_is_prefix() tests if one composite name is a
	prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to fn_composite_name_suffix() will return the remainder of the name.) Otherwise, it returns 0 and the value of the iteration marker is undefined. The function fn_composite_name_is_suffix() is similar. It tests if one composite name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.
	The functions fn_composite_name_prepend_comp() and fn_composite_name_append_comp() prepend and append a single component to the given composite name, respectively. These operations invalidate any iteration marker the client holds for that object. fn_composite_name_insert_comp() inserts a single component before <i>iter_pos</i> to the given composite name and sets <i>iter_pos</i> to be immediately after the component just inserted. fn_composite_name_delete_comp() deletes the component located before <i>iter_pos</i> from the given composite name and sets <i>iter_pos</i> back one component.
	The functions fn_composite_name_prepend_name(), fn_composite_name_append_name(), and fn_composite_name_insert_name() perform the same update functions as their _comp counterparts, respectively, except that multiple components are being added, rather than single components. For example, fn_composite_name_insert_name() sets <i>iter_pos</i> to be immediately after the name just added.
RETURN VALUES	The functions fn_composite_name_is_empty(), fn_composite_name_is_equal(), fn_composite_name_is_suffix(), and fn_composite_name_is_prefix() return 1 if the test indicated is true; 0 otherwise.
	The update functions fn_composite_name_prepend_comp(), fn_composite_name_append_comp(), fn_composite_name_insert_comp(), fn_composite_name_delete_comp(), and their _ <i>name</i> counterparts return 1 if the update was successful; 0 otherwise.
	If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.
ERRORS	Code set mismatches that occur during the composition of the string form or during comparisons of composite names are resolved in an implementation-dependent way. fn_string_from_composite_name(), fn_composite_name_is_equal(), fn_composite_name_is_suffix(), and fn_composite_name_is_prefix() set <i>status</i> to FN_E_INCOMPATIBLE_CODE_SETS for composite names whose components have code sets that are determined by the implementation to be incompatible.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

#### FN\_composite\_name\_t(3XFN)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### **SEE ALSO** FN\_string\_t(3XFN), xfn(3XFN), attributes(5)

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME	FN_compound_name_t, fn_compound_name_from_syntax_attrs, fn_compound_name_get_syntax_attrs, fn_compound_name_destroy, fn_string_from_compound_name, fn_compound_name_copy, fn_compound_name_assign, fn_compound_name_count, fn_compound_name_first, fn_compound_name_next, fn_compound_name_prev, fn_compound_name_last, fn_compound_name_prefix, fn_compound_name_suffix, fn_compound_name_is_empty, fn_compound_name_is_equal, fn_compound_name_is_prefix, fn_compound_name_is_suffix, fn_compound_name_prepend_comp, fn_compound_name_append_comp, fn_compound_name_insert_comp, fn_compound_name_delete_comp, fn_compound_name_delete_all – an XFN compound name
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_compound_name_t *fn_compound_name_from_syntax_attrs(const FN_attrset_t *aset, const FN_string_t *name, FN_status_t *status);</pre>
	<pre>FN_attrset_t *fn_compound_name_get_syntax_attrs(const</pre>
	<pre>void fn_compound_name_destroy(FN_compound_name_t *name);</pre>
	<pre>FN_string_t *fn_string_from_compound_name(const FN_compound_name_t *name);</pre>
	<pre>FN_compound_name_t *fn_compound_name_copy(const FN_compound_name_t *name);</pre>
	<pre>FN_compound_name_t *fn_compound_name_assign(FN_compound_name_t *dst, const FN_compound_name_t *src);</pre>
	<pre>unsigned int fn_compound_name_count(const FN_compound_name_t  *name);</pre>
	<pre>const FN_string_t *fn_compound_name_first(const     FN_compound_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_compound_name_next(const     FN_compound_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_compound_name_prev(const     FN_compound_name_t *name, void **iter_pos);</pre>
	<pre>const FN_string_t *fn_compound_name_last(const     FN_compound_name_t *name, void **iter_pos);</pre>
	<pre>FN_compound_name_t *fn_compound_name_prefix(const FN_compound_name_t *name, const void *iter_pos);</pre>
	<pre>FN_compound_name_t *fn_compound_name_suffix(const FN_compound_name_t *name, const void *iter_pos);</pre>

#### FN\_compound\_name\_t(3XFN)

int **fn compound name is empty**(const FN compound name t \*name);

- int fn\_compound\_name\_prepend\_comp(FN\_compound\_name\_t \*name, const FN\_string\_t \*atomic\_comp, unsigned int \*status);
- int fn\_compound\_name\_append\_comp(FN\_compound\_name\_t \*name, const FN\_string\_t \*atomic\_comp, unsigned int \*status);

- int fn\_compound\_name\_delete\_all(FN\_compound\_name\_t \*name);

**DESCRIPTION** Most applications treat names as opaque data. Hence, the majority of clients of the XFN interface will not need to parse names. Some applications, however, such as browsers, need to parse names. For these applications, XFN provides support in the form of the FN\_compound\_name\_t object.

Each naming system in an XFN federation potentially has its own naming conventions. The FN\_compound\_name\_t object has associated operations for applications to process compound names that conform to the XFN model of expressing compound name syntax. The XFN syntax model for compound names covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention. See xfn compound names(3XFN).

An FN\_compound\_name\_t object is constructed by the operation fn\_compound\_name\_from\_syntax\_attrs, using a string name and an attribute set containing the "fn\_syntax\_type" (with identifier format FN\_ID\_STRING) attribute identifying the namespace syntax of the string name. The value "standard" (with identifier format FN\_ID\_STRING) in the "fn\_syntax\_type" specifies a syntax model that is by default supported by the FN\_compound\_name\_t object. An implementation may support other syntax types instead of the XFN standard syntax model, in which case the value of the "fn\_syntax\_type" attribute would be set to an implementation-specific string. fn\_compound\_name\_get\_syntax\_attrs() returns an attribute set containing the syntax attributes that describes the given compound name. fn\_compound\_name\_destroy() releases the storage associated with the given compound name. fn\_string\_from\_compound\_name() returns the string

#### FN\_compound\_name\_t(3XFN)

form of the given compound name. fn\_compound\_name\_copy() returns a copy of the given compound name. fn\_compound\_name\_assign() makes a copy of the compound name *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the object pointed to by *dst* is returned. fn\_compound\_name\_count() returns the number of atomic components in the given compound name.

The function fn\_compound\_name\_first() returns a handle to the FN\_string\_t that is the first atomic component in the compound name, and sets *iter\_pos* to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the fn\_compound\_name\_next() function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, fn\_compound\_name\_next() returns 0. Similarly, fn\_compound\_name\_prev() returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, fn\_compound\_name\_prev() returns 0. The function fn\_compound\_name\_last() returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to fn\_compound\_name\_prev() can be used to step through trailing components of the name).

The fn\_compound\_name\_suffix() function returns a compound name consisting of a copy of those components following the supplied iteration marker. The function fn\_compound\_name\_prefix() returns a compound name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized with the use of fn\_compound\_name\_first(), fn\_compound\_name\_last(), fn\_compound\_name\_is\_prefix(), or fn\_compound\_name\_is\_suffix() yields undefined and generally undesirable behavior.

The functions fn compound name is equal(),

fn\_compound\_name\_is\_prefix(), and fn\_compound\_name\_is\_suffix() test for equality between compound names or between parts of compound names. For these functions, equality is defined as name equivalence. A name's syntactic property, such as case-insensitivity, is taken into account by these functions.

The function fn\_compound\_name\_is\_prefix() tests if one compound name is a prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to fn\_compound\_name\_suffix() will return the remainder of the name.) Otherwise, it returns 0 and value of the iteration marker is undefined. The function fn\_compound\_name\_is\_suffix() is similar. It tests if one compound name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.

The functions fn\_compound\_name\_prepend\_comp() and fn\_compound\_name\_append\_comp() prepend and append a single atomic component to the given compound name, respectively. These operations invalidate

FN\_compound\_name\_t(3XFN)

	any iteration marker the client holds for that object. fn_compound_name_insert_comp() inserts an atomic component before <i>iter_pos</i> to the given compound name and sets <i>iter_pos</i> to be immediately after the component just inserted. fn_compound_name_delete_comp() deletes the atomic component located before <i>iter_pos</i> from the given compound name and sets <i>iter_pos</i> back one component. fn_compound_name_delete_all() deletes all the atomic components from <i>name</i> .		
RETURN VALUES	<b>ALUES</b> The following test functions return 1 if the test indicated is true; otherwise, th 0:		
	<pre>fn_compound_name_is_empty() fn_compound_name_is_equal() fn_compound_name_is_suffix() fn_compound_name_is_prefix()</pre>		
	The following update functions return 1 if the update was successful; otherwise, they return 0:		
	<pre>fn_compound_name_prepend_comp() fn_compound_name_append_comp() fn_compound_name_insert_comp() fn_compound_name_delete_comp() fn_compound_name_delete_all()</pre>		
	If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.		
ERRORS	When the function fn_compound_name_from_syntax_attrs() fails, it returns a status code in <i>status</i> . The possible status codes are:		
	FN_E_ILLEGAL_NAME	The name supplied to the operation was not a well- formed XFN compound name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.	
	FN_E_INCOMPATIBLE_CODE_SETS	The code set of the given string is incompatible with that supported by the compound name.	
	FN_E_INVALID_SYNTAX_ATTRS	The syntax attributes supplied are invalid or insufficient to fully specify the syntax.	
	FN_E_SYNTAX_NOT_SUPPORTED	The syntax type specified is not supported.	
	The following functions may return in <i>status</i> the status code FN_E_INCOMPATIBLE_CODE_SETS when the code set of the given string is incompatible with that of the compound name:		
	<pre>fn_compound_name_is_equal()</pre>		

<pre>fn_compound_name_is_suffix()</pre>
<pre>fn_compound_name_is_prefix()</pre>
<pre>fn_compound_name_prepend_comp()</pre>
<pre>fn_compound_name_append_comp()</pre>
<pre>fn_compound_name_insert_comp()</pre>

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_attribute\_t(3XFN), FN\_attrset\_t(3XFN), FN\_composite\_name\_t(3XFN), FN\_status\_t(3XFN), FN\_string\_t(3XFN), fn\_ctx\_get\_syntax\_attrs (3XFN), xfn(3XFN), xfn\_compound\_names(3XFN), attributes(5)

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

# fn\_ctx\_bind(3XFN)

NAME	fn_ctx_bind – bind a reference to a name		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>int fn_ctx_bind(FN_ctx_t *ctx, const FN_composite_name_t *name,</pre>		
DESCRIPTION	This operation binds the supplied reference <i>ref</i> to the supplied composite name <i>name</i> relative to <i>ctx</i> . The binding is made in the target context, that is, the context named by all but the terminal atomic part of <i>name</i> . The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.		
	The value of <i>exclusive</i> determines what happens if the terminal atomic part of the name is already bound in the target context. If <i>exclusive</i> is nonzero and <i>name</i> is already bound, the operation fails. If <i>exclusive</i> is 0, the new binding replaces any existing binding.		
<b>RETURN VALUES</b>	When the bind operation is successful it returns 1; on error it returns 0.		
ERRORS	<pre>fn_ctx_bind sets status as described in FN_status_t(3XFN) and xfn_status_codes. Of special relevance for this operation is the status code FN_E_NAME_IN_USE, which indicates that the supplied name is already in use.</pre>		
USAGE	The value of <i>ref</i> cannot be NULL. If the intent is to reserve a name using fn_ctx_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference defined in the X/Open registry (see fns_references(5)).		
	If multiple sources are updating a reference, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	<pre>FN_composite_name_t(3XFN), FN_ctx_t(3XFN), FN_ref_t(3XFN), FN_status_t(3XFN), fn_ctx_lookup(3XFN), fn_ctx_unbind(3XFN), xfn(3XFN), xfn_status_codes(3XFN), attributes(5), fns_references(5)</pre>		
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of		

fn\_ctx\_bind(3XFN)

Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

fn\_ctx\_create\_subcontext(3XFN)

NAME	fn_ctx_create_subcontext – create a subcont	ext in a context	
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_ref_t *fn_ctx_create_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>		
DESCRIPTION	This operation creates a new XFN context of the same type as the target context — that named by all but the terminal atomic component of <i>name</i> — and binds it to the supplied composite name.		
	As with fn_ctx_bind(), the target context must already exist. The new context is created and bound in the target context using the terminal atomic name in <i>name</i> . The operation returns a reference to the newly created context.		
RETURN VALUE	fn_ctx_create_subcontext() returns a reference to the newly created context; if the operation fails, it returns a NULL pointer (0).		
ERRORS	<pre>fn_ctx_create_subcontext() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). Of special relevance for this operation is the following status code:</pre>		
	FN_E_NAME_IN_USE       The terminal atomic name already exists in the target context.		
APPLICATION USAGE	The new subcontext is an XFN context and is created in the same naming system as the target context. The new subcontext also inherits the same syntax attributes as the target context. XFN does not specify any further properties of the new subcontext. The target context and its naming system determine these.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Safe.	
SEE ALSO	<pre>FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), fn_ctx_bind(3XFI fn_ctx_destroy_subcontext(3XFN), x attributes(5)</pre>	N), fn_ctx_lookup(3XFN),	
	I		

104 man pages section 3: Networking Library Functions • Last Revised 30 Dec 1996

NAME	fn_ctx_destroy_subcontext – destroy the named context and remove its binding from the parent context			
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>			
	<pre>int fn_ctx_destroy_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>			
DESCRIPTION	This operation destroys the subcontext named by <i>name</i> relative to <i>ctx</i> , and unbinds the name.			
			eeds even if the terminal atomic ext named by all but the terminal	
<b>RETURN VALUE</b>	fn_ctx_destroy_subcontext() returns 1 on success and 0 on failure.			
ERRORS	<pre>fn_ctx_destroy_subcontext() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). Of special relevance for fn_ctx_destroy_subcontext() are the following status codes:</pre>			
	FN_E_CTX_NOT_A_CONTEXT <i>name</i> does not name a context.			
	FN_E_CTX_NOT_EMPTY		being asked to do the destroy does of a context that still contains	
APPLICATION USAGE	Some aspects of this operation are not specified by XFN, but are determined by the target context and its naming system. For example, XFN does not specify what happens if the named subcontext is non-empty when the operation is invoked.			
	In naming systems that support attributes, and store the attributes along with names or contexts, this operation removes the name, the context, and its associated attributes.			
	Normal resolution always follows links. In a fn_ctx_destroy_subcontext() operation, resolution of <i>name</i> continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation fails with FN_E_CTX_NOT_A_CONTEXT because the name is not bound to a context.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ATTRIBUTE TYPE		ATTRIBUTE VALUE	
	MT-Level	Safe.		
SEE ALSO	<pre>FN_ctx_t(3XFN), FN_compo fn_ctx_create_subconte: xfn_status_codes(3XFN),</pre>	kt(3XFN),fn_ctx_u		

## fn\_ctx\_equivalent\_name(3XFN)

NAME	fn_ctx_equivalent_name - construct an equivalent name in same context		
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_composite_name_t *fn_ctx_equivalent_name(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_string_t *leading_name, FN_status_t * status);</pre>		
DESCRIPTION	Given the name of an object <i>name</i> relative to the context <i>ctx</i> , this operation returns an equivalent name for that object, relative to the same context <i>ctx</i> , that has <i>leading_name</i> as its initial atomic name. Two names are said to be equivalent if they have prefixes that resolve to the same context, and the parts of the names immediately following the prefixes are identical.		
	The existence of a binding for <i>leading_name</i> in <i>ctx</i> does not guarantee that a name equivalent to <i>name</i> can be constructed. The failure may be because such equivalence is not meaningful, or due to the inability of the system to construct a name with the equivalence. For example, supplying _thishost as <i>leading_name</i> when <i>name</i> starts with _myself to fn_ctx_equivalent_name() in the Initial Context would not be meaningful; this results in the return of the error code FN_E_NO_EQUIVALENT_NAME.		
RETURN VALUES	If an equivalent name cannot be constructed, the value 0 is returned and <i>status</i> is set appropriately.		
ERRORS	<pre>fn_ctx_equivalent_name() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). The following status code is especially relevant for this operation:</pre>		
	FN_E_NO_EQUIVALENT_NAME	No equivalent name can be constructed, either because there is no meaningful equivalence between <i>name</i> and <i>leading_name</i> , or the system does not support constructing the requested equivalent name, for implementation-specific reasons.	
EXAMPLES	EXAMPLE 1 Naming Files		
	<pre>In the Initial Context supporting XFN enterprise policies, a user jsmith is able to name one of her files relative to this context in several waysmyself/_fs/map.psuser/jsmith/_fs/map.psorgunit/finance/_user/jsmith/_fs/map.ps The first of these may be appealing to the user jsmith in her day-to-day operations. This name is not, however, appropriate for her to use when referring the file in an electronic mail message sent to a colleague. The second of these names would be appropriate if the colleague were in the same organizational unit, and the third appropriate for anyone in the same enterprise.</pre>		

106 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

EXAMPLE 1 Naming Files (Continued) When the following sequence of instructions is executed by the user jsmith in the organizational unit finance, enterprise\_wide\_name would contain the composite name orgunit/finance/ user/jsmith/ fs/map.ps:

FN\_string\_t\* namestr =
 fn\_string\_from\_str((const unsigned char\*)"\_myself/\_fs/map.ps");
FN\_composite\_name\_t\* name = fn\_composite\_name\_from\_string(namestr);
FN\_string\_t\* org\_lead =
 fn\_string\_from\_str((const unsigned char\*)"\_orgunit");
FN\_status\_t\* status = fn\_status\_create();
FN\_composite\_name\_t\* enterprise\_wide\_name;
FN\_ctx\_t\* init\_ctx = fn\_ctx\_handle\_from\_initial(status);
/\* check status of from\_initial() \*/
enterprise\_wide\_name = fn\_ctx\_equivalent\_name(init\_ctx, name, org\_lead, status);

When the following sequence of instructions is executed by the user jsmith in the organizational unit finance, shortest\_name would contain the composite name \_myself/\_fs/map.ps:

```
FN_string_t* namestr =
    fn_string_from_str((const unsigned char*)
        "_orgunit/finance/_user_jsmith/_fs/map.ps");
FN_composite_name_t* name = fn_composite_name_from_string(namestr);
FN_string_t* mylead = fn_string_from_str((const unsigned char*)"_myself");
FN_status_t* status = fn_status_create();
FN_composite_name_t* shortest_name;
FN_ctx_t* init_ctx = fn_ctx_handle_from_initial(status);
/* check status of from_initial() */
shortest_name = fn_ctx_equivalent_name(init_ctx, name, mylead, status);
```

```
ATTRIBUTES
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE	
MT-Level	MT-Safe	

SEE ALSO FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_status\_t(3XFN), FN string t(3XFN), xfn status codes(3XFN), attributes(5)

## fn\_ctx\_get\_ref(3XFN)

fn_ctx_get_ref – return a context's reference		
<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
<pre>FN_ref_t *fn_ctx_get_ref(const FN_ctx_t *ctx, FN_status_t *status);</pre>		
This operation returns a reference to the su	ipplied context object.	
<pre>fn_ctx_get_ref() returns a pointer to an FN_ref_t object if the operation succeeds, it returns 0 if the operation fails.</pre>		
<pre>fn_ctx_get_ref() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). The following status code is of particular relevance to this operation:</pre>		
	Using the fn_ctx_get_ref() operation on the Initial Context returns this status code.	
<pre>fn_ctx_get_ref() cannot be used on the Initial Context. fn_ctx_get_ref() can be used on contexts bound in the Initial Context (in other words, the bindings in the Initial Context have references).</pre>		
If the context handle was created earlier using the fn_ctx_handle_from_ref() operation, the reference returned by the fn_ctx_get_ref() operation may not necessarily be exactly the same in content as that originally supplied. For example, fn_ctx_handle_from_ref() may construct the context handle from one address from the list of addresses. The context implementation may return with a call to fn_ctx_get_ref() only that address, or a more complete list of addresses than what was supplied in fn_ctx_handle_from_ref().		
See attributes (5) for descriptions of the following attributes:		
ATTRIBUTE TYPE	ATTRIBUTE VALUE	
MT-Level	Safe.	
<pre>FN_ctx_t(3XFN), FN_ref_t(3XFN), FN_status_t(3XFN), fn_ctx_handle_from_initial(3XFN), fn_ctx_handle_from_ref(3XFN), xfn_status_codes (3XFN), xfn(3XFN), attributes(5)</pre>		
	<pre>cc [ flag ] filelxfn [ library #include <xfn xfn.h=""> FN_ref_t *fn_ctx_get_ref (const F This operation returns a reference to the su fn_ctx_get_ref () returns a pointer to a succeeds, it returns 0 if the operation fails. fn_ctx_get_ref () sets status as describ xfn_status_codes(3XFN). The followin this operation: FN_E_OPERATION_NOT_SUPPORTED fn_ctx_get_ref () cannot be used on th be used on contexts bound in the Initial Co Initial Context have references). If the context handle was created earlier us operation, the reference returned by the fr necessarily be exactly the same in content a fn_ctx_get_ref () only that address, on what was supplied in fn_ctx_handle_f See attributes (5) for descriptions of th MT-Level FN_ctx_t(3XFN), FN_ref_t(3XFN), FN_ fn_ctx_handle_from_initial(3XFN)</xfn></pre>	

108 man pages section 3: Networking Library Functions • Last Revised 30 Dec 1996

NAME	fn_ctx_get_syntax_attrs – return syntax attr	ibutes associated with named context	
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>		
DESCRIPTION	Each context has an associated set of syntax-related attributes. This operation returns the syntax attributes associated with the context named by <i>name</i> relative to the context <i>ctx</i> .		
	The attributes must contain the attribute fn_syntax_type (FN_ID_STRING format). If the context supports a syntax that conforms to the XFN standard syntax model, fn_syntax_type is set to "standard" (ASCII attribute syntax) and the attribute set contains the rest of the relevant syntax attributes described in xfn_compound_names(3XFN).		
	This operation is different from other XFN attribute operations in that these syntax attributes could be obtained directly from the context. Attributes obtained through other XFN attribute operations may not necessarily be associated with the context; they may be associated with the reference of context, rather than the context itself (see xfn_attributes(3XFN)).		
RETURN VALUE	<pre>fn_ctx_get_syntax_attrs() returns an attribute set if successful; it returns a NULL pointer (0) if the operation fails.</pre>		
ERRORS	<pre>fn_ctx_get_syntax_attrs() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).</pre>		
APPLICATION USAGE	Implementations may choose to support other syntax types in addition to, or in place of, the XFN standard syntax model, in which case, the value of the fn_syntax_type attribute would be set to an implementation-specific string, and different or additional syntax attributes will be in the set. Syntax attributes of a context may be generated automatically by a context, in response to fn ctx get syntax attrs(), or they may be created and updated		
	using the base attribute operations. This is implementation-dependent.		
ATTRIBUTES	See attributes (5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level	Safe.	
	INIT-LEVEI	Salt.	
SEE ALSO	<pre>FN_attrset_t(3XFN), FN_composite_r FN_compound_name_t(3XFN), FN_ctx_t fn_attr_get(3XFN), fn_attr_multi_g</pre>	(3XFN), FN_status_t(3XFN),	

fn\_ctx\_get\_syntax\_attrs(3XFN)

xfn_ xfn_	_compoun _status	d_names codes(3X	(3XFN), : (FN), xfi	n(3XFN),	attrik	s(3XFN outes(5

**NAME** | fn\_ctx\_handle\_destroy – release storage associated with context handle

SYNOPSIS cc [ flag ... ] file ... -lxfn [ library ... ] #include <xfn/xfn.h>

void fn\_ctx\_handle\_destroy(FN\_ctx\_t \*ctx);

**DESCRIPTION** This operation destroys the context handle *ctx* and allows the implementation to free resources associated with the context handle. This operation does not affect the state of the context itself.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SEE ALSO FN\_ctx\_t(3XFN), fn\_ctx\_handle\_from\_initial(3XFN), fn\_ctx\_handle\_from\_ref(3XFN), xfn(3XFN), attributes(5)

## fn\_ctx\_handle\_from\_initial(3XFN)

NAME	fn_ctx_handle_from_initial – return a handle to the Initial Context	
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library #include <xfn xfn.h=""></xfn></pre>	]
	<pre>FN_ctx_t *fn_ctx_handle_from_initial(unsigned int authoritative, FN_status_t *status);</pre>	
DESCRIPTION	This operation returns a handle to the caller's Initial Context. On successful return, the handle points to a context which meets the specification of the XFN Initial Context (see fns_initial_context(5)).	
	<i>authoritative</i> specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When <i>authoritative</i> is 0, the handle to the context returned need not be authoritative.	
RETURN VALUES	<pre>fn_ctx_handle_from_initial() returns a pointer to an FN_ctx_t object if the operation succeeds; it returns a NULL pointer (0) otherwise.</pre>	
ERRORS	<pre>fn_ctx_handle_from_initial() sets only the status code portion of the status object status.</pre>	
USAGE	Authoritativeness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.	
	Applications set <i>authoritative</i> to 0 for typical day-to-day operations. Applications only set <i>authoritative</i> to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.	
	It is implementation-dependent whether authoritativeness is transferred from one context to the next as composite name resolution proceeds. Getting an authoritative context handle to the Initial Context means that operations on bindings in the Initial Context are processed using the most authoritative information. Contexts referenced implicitly through an authoritative Initial Context (for example, through the use of composite names) may not necessarily themselves be authoritative.	
ATTRIBUTES	See attributes (5) for descriptions of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe

112 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

fn\_ctx\_handle\_from\_initial(3XFN)

- SEE ALSO | FN\_ctx\_t(3XFN), FN\_status\_t(3XFN), fn\_ctx\_get\_ref(3XFN), fn\_ctx\_handle\_from\_ref(3XFN), xfn(3XFN), xfn\_status\_codes(3XFN), attributes(5), fns\_initial\_context(5)
  - **NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

## fn\_ctx\_handle\_from\_ref(3XFN)

NAME	fn_ctx_handle_from_ref – construct a handle to a context object using the given reference		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, unsigned int</pre>		
DESCRIPTION	This operation creates a handle to an FN_ctx_t object using an FN_ref_t object for that context.		
	<i>authoritative</i> specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When <i>authoritative</i> is 0, the handle to the context returned need not be authoritative.		
RETURN VALUES	This operation returns a pointer to an FN_ctx_t object if the operation succeeds; otherwise, it returns a NULL pointer (0).		
ERRORS	<pre>fn_ctx_handle_from_ref() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). The following status code is of particular relevance to this operation:</pre>		
	FN_E_NO_SUPPORTED_ADDRESS A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.		
USAGE	Authoritativeness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.		
	Applications set <i>authoritative</i> to 0 for typical day-to-day operations. Applications only set <i>authoritative</i> to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.		
	To control the authoritativeness of the target context, the application first resolves explicitly to the target context using fn_ctx_lookup(3XFN). It then uses fn_ctx_handle_from_ref() with the appropriate authoritative argument to obtain a handle to the context. This returns a handle to a context with the specified authoritativeness. The application then uses the XFN operations, such as lookup and list, with this context handle.		

### fn\_ctx\_handle\_from\_ref(3XFN)

It is implementation-dependent whether authoritativeness is transferred from one context to the next as composite name resolution proceeds. The application should use the approach recommended above to achieve the desired level of authoritativeness on a per context basis.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_ctx\_t(3XFN), FN\_ref\_t(3XFN), FN\_status\_t(3XFN), fn\_ctx\_get\_ref(3XFN), fn\_ctx\_handle\_destroy(3XFN), fn\_ctx\_lookup(3XFN), xfn(3XFN), xfn\_status\_codes(3XFN), attributes(5), fns\_references(5)

# fn\_ctx\_list\_bindings(3XFN)

	fn_ctx_list_bindings, FN_bindinglist_t, fn_bindinglist_next, fn_bindinglist_destroy – list the atomic names and references bound in a context		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_bindinglist_t *fn_ctx_list_bindings(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>		
	<pre>FN_string_t *fn_bindinglist_next(FN_bindinglist_t *bl, FN_ref_t</pre>		
	<pre>void fn_bindinglist_destroy(FN_bindinglist_t *bl, FN_status_t  *status);</pre>		
DESCRIPTION	This set of operations is used to list the names and bindings in the context named by <i>name</i> relative to the context <i>ctx</i> . Note that <i>name</i> must name a context. If the intent is to list the contents of <i>ctx, name</i> should be an empty composite name.		
	The semantics of these operations are similar to those for listing names (see fn_ctx_list_names(3XFN)). In addition to a name string being returned, fn_bindinglist_next() also returns the reference of the binding for each member of the enumeration.		
ATTRIBUTES	See attributes (5) for descriptions of the	e following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	MT-Level FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), FN_string_t(3XFI xfn(3XFN), xfn_status_codes(3XFN), a	MT-Safe t(3XFN), FN_ref_t(3XFN), N), fn_ctx_list_names(3XFN),	

NAME	fn_ctx_list_names, FN_namelist_t, fn_namelist_next, fn_namelist_destroy – list the atomic names bound in a context	
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>	
	<pre>FN_namelist_t *fn_ctx_list_name</pre>	
	<pre>FN_string_t *fn_namelist_next(F</pre>	N_namelist_t * <i>nl</i> , FN_status_t
	void <b>fn_namelist_destroy</b> (FN_nam	<pre>nelist_t *nl, FN_status_t *status);</pre>
DESCRIPTION	This set of operations is used to list the names bound in the target context named <i>na</i> relative to the context <i>ctx</i> . Note that <i>name</i> must name a context. If the intent is to list the contents of <i>ctx</i> , <i>name</i> should be an empty composite name.	
	The call to fn_ctx_list_names() initian handle to an FN_namelist_t object that target context.	ates the enumeration process. It returns a can be used to enumerate the names in the
	The operation fn_namelist_next() returns the next name in the enumeration identified by nl and updates nl to indicate the state of the enumeration. Successive calls to fn_namelist_next() using nl return successive names in the enumeration and further update the state of the enumeration. fn_namelist_next() returns a NULL pointer (0) when the enumeration has been completed.	
	<pre>fn_namelist_destroy() is used to rel This may be invoked at any time to terming</pre>	lease resources used during the enumeration. nate the enumeration.
RETURN VALUES	<pre>fn_ctx_list_names() returns a pointer to an FN_namelist_t object if the enumeration is successfully initiated; otherwise it returns a NULL pointer (0). fn_namelist_next() returns a NULL pointer (0) if no more names can be returned in the enumeration. In the case of a failure, these operations return in <i>status</i> a code indicating the nature of the failure.</pre>	
ERRORS	Each successful call to fn_namelist_next() returns a name and sets <i>status</i> to FN_SUCCESS.	
	When fn_namelist_next() returns a NULL pointer (0), it indicates that no more names can be returned. <i>status</i> is set in the following way:	
	FN_SUCCESS	The enumeration has completed successfully.
	FN_E_INVALID_ENUM_HANDLE	The supplied enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or

Networking Library Functions 117

fn\_ctx\_list\_names(3XFN)

the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context). The enumeration is not yet complete but FN E PARTIAL RESULT cannot be continued. Other status codes, such as FN E COMMUNICATION FAILURE, are also possible in calls to fn ctx list names(), fn namelist next(), and fn namelist destroy(). These functions set *status* for these other status codes as described in FN status t(3XFN) and xfn status codes(3XFN). USAGE The names enumerated using fn namelist next() are not ordered in any way. There is no guaranteed relation between the order in which names are added to a context and the order of names obtained by enumeration. The specification does not guarantee that any two series of enumerations will return the names in the same order. When a name is added to or removed from a context, this may or may not invalidate the enumeration handle that the client holds for that context. If the enumeration handle becomes invalid, the status code FN E INVALID ENUM HANDLE is returned in status. If the enumeration handle remains valid, the update may or may not be visible to the client. In addition, there may be a relationship between the *ctx* argument supplied to fn ctx list names() and the FN namelist t object it returns. For example, some implementations may store the context handle *ctx* within the FN namelist t object for subsequent fn namelist next() calls. In general, a fn ctx handle destroy(3XFN) should not be invoked on ctx until the enumeration has terminated. EXAMPLES **EXAMPLE 1** A sample program. The following code fragment illustrates how the list names operations may be used: extern FN\_string\_t \*user\_input; FN ctx t \*ctx; FN composite name t \*target name = fn composite name from string(user input); FN status t \*status = fn status create(); FN string t \*name; FN namelist t \*nl; ctx = fn ctx handle from initial(status); /\* error checking on 'status' \*/ if ((nl=fn\_ctx\_list\_names(ctx, target\_name, status)) == 0) { /\* report 'status' and exit \*/ while (name=fn namelist next(nl, status)) { /\* do something with 'name' \*/ fn string destroy(name); /\* check 'status' for reason for end of enumeration and report if necessary \*/

```
EXAMPLE 1 A sample program. (Continued)
/* clean up */
fn namelist destroy(nl, status);
```

/\* report 'status' \*/

```
ATTRIBUTES
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO

FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_status\_t(3XFN), FN\_string\_t(3XFN), fn\_ctx\_handle\_destroy(3XFN), xfn(3XFN), xfn status codes(3XFN), attributes(5)

## fn\_ctx\_lookup(3XFN)

NAME	fn_ctx_lookup – look up name in context		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_ref_t *fn_ctx_lookup(FN_ctx_t *ctx, const FN_composite_name_t                                  *name, FN_status_t *status);</pre>		
DESCRIPTION	This operation returns the reference bound	to <i>name</i> relative to the context <i>ctx</i> .	
RETURN VALUE	If the operation succeeds, the fn_ctx_loo reference bound to <i>name</i> . Otherwise, 0 is re		
ERRORS	<pre>fn_ctx_lookup() sets status as described FN_status_t(3XFN) and xfn_status_codes(3XFN).</pre>		
APPLICATION USAGE	Some naming services may not always have reference information for all names in their contexts; for such names, such naming services may return a special reference whose type indicates that the name is not bound to any address. This reference may be name service specific or it may be the conventional NULL reference defined in the X/Open registry. See fns_references(5).		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Safe.	
SEE ALSO	MT-Level FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), fns_references( xfn(3XFN), attributes(5)	Safe. t(3XFN), FN_ref_t(3XFN),	
SEE ALSO	FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), fns_references(	Safe. t(3XFN), FN_ref_t(3XFN),	

fn_ctx_lookup_link – look up the link reference bound to a name		
<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
<pre>FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>		
This operation returns the XFN link bound to <i>name</i> . The terminal atomic part of <i>name</i> must be bound to an XFN link.		
The normal fn_ctx_lookup(3XFN) operation follows all links encountered, including any bound to the terminal atomic part of <i>name</i> . This operation differs from the normal lookup in that when the terminal atomic part of <i>name</i> is an XFN link, this link is not followed, and the operation returns the link.		
If fn_ctx_lookup_link() fails, a NULL I	pointer (0) is returned.	
<pre>fn_ctx_lookup_link() sets status as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). Of special relevance for fn_ctx_lookup_link() is the following status code:</pre>		
FN_E_MALFORMED_LINK <i>name</i> resolved to a reference that was not a link.		
See attributes(5) for descriptions of the following attributes:		
ATTRIBUTE TYPE	ATTRIBUTE VALUE	
MT-Level	MT-Safe	
MT-Level FN_composite_name_t(3XFN), FN_ctx_ FN_status_t(3XFN), fn_ctx_lookup(3) xfn_status_codes(3XFN), attributes	t(3XFN), FN_ref_t(3XFN), KFN), xfn(3XFN), xfn_links(3XFN),	
	<pre>#include <xfn xfn.h=""> FN_ref_t *fn_ctx_lookup_link(FN_G_FN_composite_name_t *name, FN This operation returns the XFN link bound must be bound to an XFN link. The normal fn_ctx_lookup(3XFN) opera including any bound to the terminal atomic the normal lookup in that when the terminal link is not followed, and the operation return If fn_ctx_lookup_link() fails, a NULL p fn_ctx_lookup_link() sets status as de xfn_status_codes(3XFN). Of special relation the following status code: FN_E_MALFORMED_LINK name resolve See attributes(5) for descriptions of the</xfn></pre>	

### fn\_ctx\_rename(3XFN)

NAME	fn_ctx_rename – rename the name of a binding		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>int fn_ctx_rename(FN_ctx_t *ctx, c *oldname, const FN_composite_r exclusive, FN_status_t *status);</pre>		
DESCRIPTION	The fn_ctx_rename() operation binds the reference currently bound to <i>oldname</i> relative to <i>ctx</i> , to the name <i>newname</i> , and unbinds <i>oldname</i> . <i>newname</i> is resolved relative to the target context (that named by all but the terminal atomic part of <i>oldname</i> ).		
	If <i>exclusive</i> is 0, the operation overwrites any old binding of <i>newname</i> . If <i>exclusive</i> is nonzero, the operation fails if <i>newname</i> is already bound.		
<b>RETURN VALUES</b>	<pre>fn_ctx_rename() returns 1 if the operation </pre>	on is successful, 0 otherwise.	
ERRORS	<pre>fn_ctx_rename() sets status as described FN_status_t(3XFN) and xfn_status_codes(3XFN).</pre>		
USAGE	The only restriction that XFN places on <i>newname</i> is that it be resolved relative to the target context. XFN does not specify further restrictions on <i>newname</i> . For example, in some implementations, <i>newname</i> might be restricted to be a name in the same naming system as the terminal component of <i>oldname</i> . In another implementation, <i>newname</i> might be restricted to be an atomic name.		
	Normal resolution always follows links. In an fn_ctx_rename() operation, resolution of <i>oldname</i> continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation binds <i>newname</i> to the link and unbinds the terminal atomic name of <i>oldname</i> .		
	In naming systems that support attributes and store the attributes along with the names, the unbind of the terminal atomic name of <i>oldname</i> also removes its associated attributes. It is implementation-dependent whether these attributes become associated with <i>newname</i> .		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	

SEE ALSO
FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_ref\_t(3XFN),
FN\_status\_t(3XFN), fn\_ctx\_bind(3XFN) fn\_ctx\_unbind(3XFN), xfn(3XFN),
xfn\_status\_codes(3XFN), attributes(5)

122 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

fn\_ctx\_rename(3XFN)

FN\_ctx\_t(3XFN)

NAME	FN_ctx_t – an XFN context
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_ctx_t *fn_ctx_handle_from_initial(unsigned int authoritative, FN_status_t *status);</pre>
	<pre>FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, unsigned int</pre>
	<pre>FN_ref_t *fn_ctx_get_ref(const FN_ctx_t *ctx, FN_status_t *status);</pre>
	<pre>void fn_ctx_handle_destroy(FN_ctx_t *ctx);</pre>
	<pre>FN_ref_t *fn_ctx_lookup(FN_ctx_t *ctx, const FN_composite_name_t                                 *name, FN_status_t *status);</pre>
	<pre>FN_namelist_t *fn_ctx_list_names(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>FN_string_t *fn_namelist_next(FN_namelist_t *nl, FN_status_t</pre>
	<pre>void fn_namelist_destroy(FN_namelist_t *nl, FN_status_t *status);</pre>
	<pre>FN_bindinglist_t *fn_ctx_list_bindings(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>FN_string_t *fn_bindinglist_next(FN_bindinglist_t *iter, FN_ref_t</pre>
	<pre>void fn_bindinglist_destroy(FN_bindinglist_t *iter_pos, FN_status_t</pre>
	<pre>int fn_ctx_bind(FN_ctx_t *ctx, const FN_composite_name_t *name,</pre>
	<pre>int fn_ctx_unbind(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>int fn_ctx_rename(FN_ctx_t *ctx, const FN_composite_name_t     *oldname, const FN_composite_name_t *newname, unsigned int     exclusive, FN_status_t *status);</pre>
	<pre>FN_ref_t *fn_ctx_create_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>int fn_ctx_destroy_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>
	<pre>FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>

124 man pages section 3: Networking Library Functions • Last Revised 13 Dec 1996

DESCRIPTION	<b>ON</b> An XFN context consists of a set of name to reference bindings. An XFN context is represented by the type FN_ctx_t in the client interface. The operations for manipulating an FN_ctx_t object are described in detail in separate reference manupages.		
The following contains a brief summary of these operations:			
	<pre>fn_ctx_handle_from_initial() returns a pointer to an Initial Context that provides a starting point for resolution of composite names. fn_ctx_handle_from_ref() returns a handle to an FN_ctx_t object using the given reference ref. fn_ctx_get_ref() returns the reference of the context ctx. fn_ctx_handle_destroy() releases the resources associated with the FN_ctx_t object ctx; it does not affect the state of the context itself.</pre>		
	<pre>fn_ctx_lookup() returns the reference bound to name resolved relative to ctx. fn_ctx_list_names() is used to enumerate the atomic names bound in the context named by name resolved relative to ctx. fn_ctx_list_bindings() is used to enumerate the atomic names and their references in the context named by name resolved relative to ctx.</pre>		
	<pre>fn_ctx_bind() binds the composite name name to a reference ref resolved relative to ctx.fn_ctx_unbind() unbinds name resolved relative to ctx.fn_ctx_rename() binds newname to the reference bound to oldname and unbinds oldname. oldname is resolved relative to ctx; newname is resolved relative to the target context.</pre>		
	<pre>fn_ctx_create_subcontext() creates a new context with the given composite name name resolved relative to ctx. fn_ctx_destroy_subcontext() destroys the context named by name resolved relative to ctx.</pre>		
	Normal resolution always follows links. fn_ctx_lookup_link() looks up <i>name</i> relative to <i>ctx</i> , following links except for the last atomic part of <i>name</i> , which must be bound to an XFN link.		
	fn_ctx_get_syntax_attrs() returns an attribute set containing attributes that describe a context's syntax. <i>name</i> must name a context.		
ERRORS	In each context operation, the caller supplies an FN_status_t object as a parameter. The called function sets this status object as described in FN_status_t(3XFN) and xfn_status_codes(3XFN).		
USAGE	In most of the operations of the base context interface, the caller supplies a context and a composite name. The supplied name is always interpreted relative to the supplied context.		
	The operation may eventually be effected on a different context called the operation's <i>target context</i> . Each operation has an initial resolution phase that conveys the operation to its target context, and the operation is then applied. The effect (but not necessarily the implementation) is that of doing a lookup on that portion of the name that		

#### FN\_ctx\_t(3XFN)

represents the target context, and then invoking the operation on the target context. The contexts involved only in the resolution phase are called *intermediate contexts*.

Normal resolution of names in context operations always follows XFN links.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_attrset\_t(3XFN), FN\_composite\_name\_t(3XFN), FN\_ref\_t(3XFN), FN\_status\_t(3XFN), fn\_ctx\_bind(3XFN), fn\_ctx\_create\_subcontext(3XFN), fn\_ctx\_destroy\_subcontext(3XFN), fn\_ctx\_get\_ref(3XFN), fn\_ctx\_get\_syntax\_attrs(3XFN), fn\_ctx\_handle\_destroy(3XFN), fn\_ctx\_handle\_from\_initial(3XFN), fn\_ctx\_handle\_from\_ref(3XFN), fn\_ctx\_list\_bindings(3XFN), fn\_ctx\_list\_names(3XFN), fn\_ctx\_lookup(3XFN), fn\_ctx\_lookup\_link(3XFN), fn\_ctx\_rename(3XFN), fn\_ctx\_unbind(3XFN), xfn(3XFN), xfn\_links(3XFN), xfn\_status\_codes(3XFN), attributes(5)

NAME	fn_ctx_unbind – unbind a name from a context		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>int fn_ctx_unbind(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre>		
DESCRIPTION	This operation removes the terminal atomic name in <i>name</i> from the the target context — that named by all but the terminal atomic part of <i>name</i> .		
	This operation is successful even if the terminal atomic name was not bound in target context, but fails if any of the intermediate names are not bound. fn_ctx_unbind() is idempotent.		
<b>RETURN VALUE</b>	The operation returns 1 if successful, and 0 otherwise.		
ERRORS	<pre>fn_ctx_unbind() sets status as described in FN_status_t and xfn_status_codes (3XFN).</pre>		
	Certain naming systems may disallow unbinding a name if the name is bound to an existing context in order to avoid orphan contexts that cannot be reached via any name. In such situations, the status code FN_E_OPERATION_NOT_SUPPORTED is returned.		
APPLICATION USAGE			
	Normal resolution always follows links. In an fn_ctx_unbind() operation, resolution of <i>name</i> continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Safe.	
SEE ALSO	LSO FN_composite_name_t(3XFN), FN_ctx_t(3XFN), FN_ref_t(3XFN), FN_status_t(3XFN), fn_ctx_bind(3XFN), fn_ctx_lookup(3XFN), xfn_status_codes(3XFN), xfn(3XFN), attributes(5)		

FN\_identifier\_t(3XFN)

NAME | FN\_identifier\_t - an XFN identifier

DESCRIPTION

Identifiers are used to identify reference types and address types in an XFN reference, and to identify attributes and their syntax in the attribute operations.

> An XFN identifier consists of an unsigned int, which determines the format of identifier, and the actual identifier, which is expressed as a sequence of octets.

The representation of this structure is defined by XFN as follows:

typedef struct { unsigned int format; size\_t length; void \*contents; } FN\_identifier\_t;

XFN defines a small number of standard forms for identifiers:

FN_ID_STRING	The identifier is an ASCII string (ISO 646).
FN_ID_DCE_UUID	The identifier is an OSF DCE UUID in string representation. (See the X/Open DCE RPC.)
FN_ID_ISO_OID_STRING	The identifier is an ISO OID in ASN.1 dot-separated integer list string format. (See the ISO ASN.1.)
FN_ID_ISO_OID_BER	The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. (See the ISO BER.)

FILES #include <xfn/xfn.h>

FN attribute t(3XFN), FN ref addr t(3XFN), FN ref t(3XFN), xfn(3XFN) SEE ALSO

NOTES The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

man pages section 3: Networking Library Functions • Last Revised 4 Nov 1994 128

NAME	FN_ref_addr_t, fn_ref_addr_create, fn_ref_addr_destroy, fn_ref_addr_copy, fn_ref_addr_assign, fn_ref_addr_type, fn_ref_addr_length, fn_ref_addr_data, fn_ref_addr_description – an address in an XFN reference
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_ref_addr_t *fn_ref_addr_create(constFN_identifier_t *type,</pre>
	<pre>void fn_ref_addr_destroy(FN_ref_addr_t *addr);</pre>
	<pre>FN_ref_addr_t *fn_ref_addr_copy(constFN_ref_addr_t *addr);</pre>
	<pre>FN_ref_addr_t *fn_ref_addr_assign(FN_ref_addr_t *dst, const FN_ref_addr_t *src);</pre>
	<pre>const FN_identifier_t *fn_ref_addr_type(constFN_ref_addr_t     *addr);</pre>
	<pre>size_t fn_ref_addr_length(const FN_ref_addr_t *addr);</pre>
	<pre>const void* fn_ref_addr_data(const FN_ref_addr_t *addr);</pre>
	<pre>FN_string_t *fn_ref_addr_description(constFN_ref_addr_t *addr, unsigned int detail, unsigned int *more_detail);</pre>
DESCRIPTION	An XFN reference is represented by the type FN_ref_t. An object of this type contains a reference type and a list of addresses. Each address in the list is represented by an object of type FN_ref_addr_t. An address consists of an opaque data buffer and a type field, of type FN_identifier_t.
	<pre>fn_ref_addr_create() creates and returns an address with the given type and data. length indicates the size of the data. fn_ref_addr_destroy() releases the storage associated with the given address. fn_ref_addr_copy() returns a copy of the given address object. fn_ref_addr_assign() makes a copy of the address pointed to by src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.</pre>
	<pre>fn_ref_addr_type() returns the type of the given address. fn_ref_addr_length() returns the size of the address in bytes. fn_ref_addr_data() returns the contents of the address.</pre>
	fn_ref_addr_description() returns the implementation-defined textual description of the address. It takes as arguments a number, <i>detail</i> , and a pointer to a number, <i>more_detail</i> . <i>detail</i> specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If <i>more_detail</i> is 0, it is ignored. If <i>more_detail</i> is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by <i>detail</i> . If no higher level of detail is available, <i>more_detail</i> is set to <i>detail</i> .

#### FN\_ref\_addr\_t(3XFN)

**USAGE** The address type of an FN\_ref\_addr\_t object is intended to identify the mechanism that should be used to reach the object using that address. The client must interpret the contents of the opaque data buffer of the address based on the type of the address, and on the type of the reference that the address is in. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

Manipulation of addresses using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to addresses in the underlying naming system can only be effected through the use of the interfaces described in FN ctx t(3XFN).

#### ATTRIBUTES See

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

### SEE ALSO FN\_ctx\_t(3XFN), FN\_identifier\_t(3XFN), FN\_ref\_t(3XFN), FN\_string\_t(3XFN), xfn(3XFN), attributes(5)

NAME	AE FN_ref_t, fn_ref_create, fn_ref_destroy, fn_ref_copy, fn_ref_assign, fn_ref_type, fn_ref_addrcount, fn_ref_first, fn_ref_next, fn_ref_prepend_addr, fn_ref_append_ fn_ref_insert_addr, fn_ref_delete_addr, fn_ref_delete_all, fn_ref_create_link, fn_ref_is_link, fn_ref_link_name, fn_ref_description – an XFN reference		
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>		
	<pre>FN_ref_t *fn_ref_create(const FN_identifier_t *ref_type);</pre>		
	<pre>void fn_ref_destroy(FN_ref_t *ref);</pre>		
	<pre>FN_ref_t *fn_ref_copy(const FN_ref_t *ref);</pre>		
	<pre>FN_ref_t *fn_ref_assign(FN_ref_t *dst, const FN_ref_t *src);</pre>		
	<pre>const FN_identifier_t *fn_ref_type(const FN_ref_t *ref);</pre>		
	unsigned int <b>fn_ref_addrcount</b> (const FN_ref_t *ref);		
	<pre>const FN_ref_addr_t *fn_ref_first(const FN_ref_t *ref, void     **iter_pos);</pre>		
	<pre>const FN_ref_addr_t *fn_ref_next(const FN_ref_t *ref, void     **iter_pos);</pre>		
	<pre>int fn_ref_prepend_addr(FN_ref_t *ref, const FN_ref_addr_t *addr);</pre>		
	<pre>int fn_ref_append_addr(FN_ref_t *ref, const FN_ref_addr_t *addr);</pre>		
	<pre>int fn_ref_insert_addr(FN_ref_t *ref, void **iter_pos, const     FN_ref_addr_t *addr);</pre>		
	<pre>int fn_ref_delete_addr(FN_ref_t *ref, void **iter_pos);</pre>		
	<pre>int fn_ref_delete_all(FN_ref_t *ref);</pre>		
	<pre>FN_ref_t *fn_ref_create_link(const FN_composite_name_t</pre>		
	<pre>int fn_ref_is_link(const FN_ref_t *ref);</pre>		
	<pre>FN_composite_name_t *fn_ref_link_name(const FN_ref_t *link_ref);</pre>		
	<pre>FN_string_t *fn_ref_description(const FN_ref_t *ref, unsigned int</pre>		
DESCRIPTION	An XFN reference is represented by the type FN_ref_t. An object of this type contains a reference type and a list of addresses. The ordering in this list at the time of binding might not be preserved when the reference is returned upon lookup.		
	The reference type is represented by an object of type FN_identifier_t. The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.		
	Each address is represented by an object of type FN_ref_addr_t.		

	<pre>fn_ref_create() creates a reference with no address, using ref_type as its reference type. Addresses can be added later to the reference using the functions described below. fn_ref_destroy() releases the storage associated with ref. fn_ref_copy() creates a copy of ref and returns it. fn_ref_assign() creates a copy of src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.</pre>
	<pre>fn_ref_addrcount() returns the number of addresses in the reference ref.</pre>
	<pre>fn_ref_first() returns the first address in ref and sets iter_pos to be after the address. It returns 0 if there is no address in the list. fn_ref_next() returns the address following iter_pos in ref and sets iter_pos to be after the address. If the iteration marker iter_pos is at the end of the sequence, fn_ref_next() returns 0.</pre>
	<pre>fn_ref_prepend_addr() adds addr to the front of the list of addresses in ref. fn_ref_append_addr() adds addr to the end of the list of addresses in ref. fn_ref_insert_addr() adds addr to ref before iter_pos and sets iter_pos to be immediately after the new reference added. fn_ref_delete_addr() deletes the address located before iter_pos in the list of addresses in ref and sets iter_pos back one address. fn_ref_delete_all() deletes all addresses in ref.</pre>
	<pre>fn_ref_create_link() creates a reference using the given composite name link_name as an address. fn_ref_is_link() tests if ref is a link. It returns 1 if it is; 0 if it is not. fn_ref_link_name() returns the composite name stored in a link reference. It returns 0 if link_ref is not a link.</pre>
	fn_ref_description() returns a string description of the given reference. It takes as argument an integer, <i>detail</i> , and a pointer to an integer, <i>more_detail</i> . <i>detail</i> specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If <i>more_detail</i> is 0, it is ignored. If <i>more_detail</i> is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by <i>detail</i> . If no higher level of detail is available, <i>more_detail</i> is set to <i>detail</i> .
<b>RETURN VALUES</b>	The following operations return 1 if the operation succeeds, 0 if the operation fails:
	<pre>fn_ref_prepend_addr() fn_ref_append_addr() fn_ref_insert_addr() fn_ref_delete_addr() fn_ref_delete_all()</pre>
USAGE	The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.
	Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may

arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

The client must interpret the contents of a reference based on the type of the addresses and the type of the reference. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Manipulation of references using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to references in the underlying naming system can only be effected through the use of the interfaces described in FN\_ctx\_t(3XFN).

#### **ATTRIBUTES**

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_composite\_name\_t(3XFN), FN\_ctx\_t(3XFN), FN\_identifier\_t(3XFN), FN\_ref\_addr\_t(3XFN), FN\_string\_t(3XFN), fn\_ctx\_lookup(3XFN), fn\_ctx\_lookup\_link(3XFN), xfn(3XFN), xfn\_links(3XFN), attributes(5)

## FN\_search\_control\_t(3XFN)

NAME	FN_search_control_t, fn_search_control_create, fn_search_control_destroy, fn_search_control_copy, fn_search_control_assign, fn_search_control_scope, fn_search_control_follow_links, fn_search_control_max_names, fn_search_control_return_ref, fn_search_control_return_attr_ids – options for attribute search		
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>		
		<pre>control_create(unsigned int scope, gned int max_names, unsigned int *return_attr_ids, unsigned int</pre>	
	<pre>void fn_search_control_destroy(FN_search_control_t *score)</pre>		
	<pre>control_copy(const );</pre>		
	<pre>FN_search_control_t     *fn_search_control_assign()     FN_search_control_t *src);</pre>	FN_search_control_t * <i>dst</i> , const	
	<pre>unsigned int fn_search_control_     *scontrol);</pre>	<pre>scope(const FN_search_control_t</pre>	
	<pre>follow_links(const );</pre>		
	<pre>unsigned int fn_search_control_max_names(const FN_search_control_t *scontrol);</pre>		
	<pre>unsigned int fn_search_control_return_ref(const FN_search_control_t *scontrol); const FN_attrset_t *fn_search_control_return_attr_ids(const FN_search_control_t *scontrol);</pre>		
DESCRIPTION	The FN_search_control_t object is used to specify options for the attribute search operation fn_attr_ext_search(3XFN).		
<pre>fn_search_control_create() creates an FN_search_control_t object information in scope, follow_links, max_names, return_ref, and return_attr_ids to search options. If the operation succeeds, fn_search_control_create() pointer to an FN_search_control_t object; otherwise, it returns a NULL pointer to an FN_search, scope, is either the named object, the named context, for named context and its subcontexts, or the named context and a context implementation defined set of subcontexts. The values for scope are:</pre>		nes, return_ref, and return_attr_ids to set the fn_search_control_create() returns a	
		named context and a context	
	FN_SEARCH_NAMED_OBJECT	Search just the given named object.	
FN_SEARCH_ONE_CONTEXT Search just the		Search just the given context.	
	FN_SEARCH_SUBTREE	Search given context and all its subcontexts.	

#### FN\_search\_control\_t(3XFN)

FN\_SEARCH\_CONSTRAINED\_SUBTREE Search given context and its subcontexts as constrained by the context-specific policy in place at the named context.

follow\_links further defines the scope and nature of the search. If follow\_links is
nonzero, the search follows XFN links. If follow\_links is 0, XFN links are not followed.
See fn\_attr\_ext\_search(3XFN) for more detail about how XFN links are treated.

max\_names specifies the maximum number of names to return in an
FN\_ext\_searchlist\_t(3XFN) enumeration (see fn\_attr\_ext\_search(3XFN)).
The names of all objects whose attributes satisfy the filter are returned when
max\_names is 0.

If *return\_ref* is non-zero, the reference bound to the named object is returned with the object's name by fn\_ext\_searchlist\_next(3XFN) (see fn\_attr\_ext\_search(3XFN)). If *return\_ref* is 0, the reference is not returned.

Attribute identifiers and values associated with named objects that satisfy the filter may be returned by fn\_ext\_searchlist\_next(3XFN). The attributes returned are those listed in *return\_attr\_ids*. If the value of *return\_attr\_ids* is 0, all attributes are returned. If *return\_attr\_ids* is an empty FN\_attrset\_t object (see FN\_attrset\_t(3XFN)), no attributes are returned. Any attribute values in *return\_attr\_ids* are ignored; only the attribute identifiers are relevant for this operation.

fn\_attr\_ext\_search(3XFN) interprets a value of 0 for the search control argument
as a default search control which has the following option settings:

scope	FN_SEARCH_ONE_CONTEXT
follow_links	0 (do not follow links)
max_names	0 (return all named objects that match filter)
return_ref	0 (do not return the reference of the named object)
return_attr_ids	an empty FN_attrset_t object (do not return any attributes of the named object)

fn\_search\_control\_destroy() releases the storage associated with scontrol.

fn\_search\_control\_copy() returns a copy of the search control scontrol.

fn\_search\_control\_assign() makes a copy of the search control *src* and assigns it to *dst*, releasing the old contents of *dst*. A pointer to the same object as *dst* is returned.

fn search control scope() returns the scope for the search.

fn\_search\_control\_follow\_links() returns non-zero if links are followed; 0 if
not.

fn\_search\_control\_max\_names() returns the maximum number of names.

Networking Library Functions 135

### FN\_search\_control\_t(3XFN)

	<pre>fn_search_control_return_ref() returns nonzero if the reference is returne if not.</pre>		
	fn_search_control_return_attr_ids() returns a pointer to the list of attributes; a NULL pointer indicates that all attributes and values are returned.		
ERRORS	<b>S</b> fn_search_control_create() returns a NULL pointer if the operation fails and sets status as follows:		
	FN_E_SEARCH_INVALID_OPTION	A supplied search option was invalid or inconsistent.	
	Other status codes are possible (see xfn_status_codes(3XFN)).		
ATTRIBUTES	See attributes(5) for descriptions of th	e following attributes:	

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO FN\_attrset\_t(3XFN), fn\_attr\_ext\_search(3XFN), xfn\_status\_codes(3XFN), attributes(5)

136 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

NAME	FN_search_filter_t, fn_search_filter_create, fn_search_filter_destroy, fn_search_filter_copy, fn_search_filter_assign, fn_search_filter_expression, fn_search_filter_arguments – filter expression for attribute search	
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>	
	<pre>FN_search_filter_t *fn_search_filter_create(unsigned int *status,</pre>	
	<pre>void fn_search_filter_destroy(FN_search_filter_t *sfilter);</pre>	
	<pre>FN_search_filter_t *fn_search_filter_copy(const</pre>	
	<pre>FN_search_filter_t *fn_search_filter_assign(FN_search_filter_t      *dst, const FN_search_filter_t *src);</pre>	
	<pre>const char *fn_search_filter_expression(const FN_search_filter_t  *sfilter);</pre>	
	<pre>const void **fn_search_filter_arguments(const FN_search_filter_t     *sfilter, size_t *number_of_arguments);</pre>	
DESCRIPTION	The FN_search_filter_t type is an expression that is evaluated against the attributes of named objects bound in the scope of the search operation fn_attr_ext_search(3XFN). The filter evaluates to TRUE or FALSE. If the filter is empty, it evaluates to TRUE. Names of objects whose attribute values satisfy the filter expression are returned by the search operation.	
	If the identifier in any subexpression of the filter does not exist as an attribute of an object, then the innermost logical expression containing that identifier is FALSE. A subexpression that is only an attribute tests for the presence of the attribute; the subexpression evaluates to TRUE if the attribute has been defined for the object and FALSE otherwise.	
	<pre>fn_search_filter_create() creates a search filter from the expression string estr and the remaining arguments.</pre>	
	<pre>fn_search_filter_destroy() releases the storage associated with the search filter sfilter.</pre>	
	<pre>fn_search_filter_copy() returns a copy of the search filter sfilter.</pre>	
	fn_search_filter_assign() makes a copy of the search filter <i>src</i> and assigns it to <i>dst</i> , releasing the old contents of <i>dst</i> . A pointer to the same object as <i>dst</i> is returned.	
	fn_search_filter_expression() returns the filter expression of <i>sfilter</i> .	
	fn_search_filter_arguments() returns an array of pointers to arguments supplied to the filter constructor. <i>number_of_arguments</i> is set to the size of this array. The types of the arguments are determined by the substitution tokens in the expression in <i>sfilter</i> .	
I		

	<pre><pilterexpr> : : = [ <expr> ] <expr> : : = <expr> "or" <expr></expr></expr></expr></expr></pilterexpr></pre>	
	Token	Argument Type
	%a	FN attribute t*
	%v	FN_attrvalue_t*
	%s	FN_string_t*
	%i	FN_identifier_t*
Precedence	The following precedence relations hold in the absence of parentheses, in the order of lowest to highest:	
	or	

		FN_search_filter_t(3XFN)	
	and not relational operators		
	These boolean and relational operators are left associative.		
Relational Operators			
	Locale (code set, language, or territory) mismatches that occur during string comparisons and ordering operations are resolved in an implementation-dependent way. Relational operations that have ordering semantics may be used for strings of code sets in which ordering is meaningful, but is not of general use in internationalized environments.		
	An attribute that occurs in the absence of any relational operator tests for the presence of the attribute.		
	Operator Meaning		
	==	The sub-expression is TRUE if at least one value of the specified attribute is equal to the supplied value.	
	! =	The sub-expression is TRUE if no values of the specified attribute equal the supplied value.	
	> =	The sub-expression is TRUE if at least one value of the attribute is greater than or equal to the supplied value.	
	>	The sub-expression is TRUE if at least one value of the attribute is greater then the supplied value.	
	< =	The sub-expression is TRUE if at least one value of the attribute is less than or equal to the supplied value.	
	<	The sub-expression is TRUE if at least one value of the attribute is less than the supplied value.	
	≈ =	The sub-expression is TRUE if at least one value of the specified attribute matches the supplied value according to some context-specific approximate matching criterion. This criterion must subsume strict equality.	
Wildcarded Strings	strings. The sequence can with either a wildcard sp	sists of a sequence of alternating wildcard specifiers and n start with either a wildcard specifier or a string, and end becifier or a string.	

The wildcard specifier is denoted by the asterisk character (' $\star$ ') and means zero or more occurrences of any character.

Wildcarded strings can be used to specify substring matches. The following are examples of wildcarded strings and what they mean:

	Wildcarded String		Meaning
	*	Any string	
	*'ing'	Any string end	ing with ing
	Any string starting		
	with jo, and containing the substring ph,		
	and which contains the substring ne in the portion		
	of the string following ph, and which ends with er		
	T }		
	%S*	Any string star	ting with the supplied string
	Any string starting with bix and ending with the supplied string		
	T}		
	String matches involving stare resolved in an impleme		ent locales (code set, language, or territory) dent way.
Extended Operations			
			ng a string, the string is used to construct an atifiers of extended operators and signatures
	The following three extend	ed operations	are currently defined:
	'name'(< <i>Wildcarded</i> Stri	ng>)	The identifier for this operation is 'name' (FN_ID_STRING). The argument to this operation is a wildcard string. The operation returns TRUE if the name of the object matches the supplied wildcard string.

'reftype'(%i)	The identifier for this operation is 'reftype' (FN_ID_STRING). The argument to this operation is an identifier. The operation returns TRUE if the reference type of the object is equal to the supplied identifier.
'addrtype'(%i)	The identifier for this operation is 'addrtype' (LM FN_ID_STRING). The argument to the operation is an identifier. The operation returns TRUE if any of the address types in the reference of the object is equal to the supplied identifier.

Support and exact semantics of extended operations are context-specific. If a context does not support an extended operation, or if the filter expression supplies the extended operation with either an incorrect number or type of arguments, the error FN\_E\_SEARCH\_INVALID\_OP is returned. (Note: FN\_E\_OPERATION\_NOT\_SUPPORTED is returned when fn\_attr\_ext\_search(3XFN) is not supported.)

The following are examples of filter expressions that contain extended operations:

Expression	Meaning
Evaluates to	
TRUE	
if the name of the object starts with	
bill.	
Τ}	
%i(%a, %v)	Evaluates to result of applying the specified operation to the supplied arguments.
(%a == %v) and 'name'('joe'*)	Evaluates to TRUE if the specified attribute has the given value and if the name of the object starts with joe.

**RETURN VALUES** fn\_search\_filter\_create() returns a pointer to an FN\_search\_filter\_object if the operation succeeds; otherwise it returns a NULL pointer.

ERRORSfn\_search\_filter\_create() returns a NULL pointer if the operation fails and sets<br/>status in the following way:FNESEARCH INVALID FILTERThe filter expression had a syntax error or

some other problem.

Networking Library Functions 141

earch_filter_t(3	XFN)	
	FN_E_SEARCH_INVALID_OP	An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation.
	FN_E_INVALID_ATTR_IDENTIFIER	The left hand side of an operator expression was not an attribute.
	FN_E_INVALID_ATTR_VALUE	The right hand side of an operator expression was not an integer, attribute value, or (wildcarded) string.
	Other status codes are possible as describ FN_status_t(3XFN) and xfn_status_	
EXAMPLES	<b>EXAMPLE 1</b> Creating Different Filters	
	The following examples illustrate how to	create three different filters.
	literals in the same filter expression. This whose color attribute contains a string	t a filter involving substitution tokens and example creates a filter for named objects value of red, blue, or white. The first two kens; the last value, white, is specified as a
	<pre>unsigned int status; extern FN_attribute_t *attr_color; FN_string_t *red = fn_string_from_str( FN_string_t *blue = fn_string_from_str FN_search_filter_t *sfilter; sfilter = fn_search_filter_create( &amp;status, "(%a == %s) or (%a == %s) or (%a == attr_color, red, attr_color, blue, attr_color);</pre>	((unsigned char *)"blue");
	-	nstruct a filter involving a wildcarded string. for named objects whose <i>last_name</i> attribute m.
	<pre>unsigned int status; extern FN_attribute_t *attr_last_name; FN_search_filter_t *sfilter; sfilter = fn_search_filter_create( &amp;status, "%a == 'm'*", attr_last_name;</pre>	ame);
		struct a filter involving extended operations. Il named objects whose name ends with ton.
	<pre>unsigned int status; FN_search_filter_t *sfilter; sfilter= fn_search_filter_create(&amp;statu</pre>	1s, "'name'(*'ton')");

142 man pages section 3: Networking Library Functions • Last Revised 22 Nov 1996

	EXAMPLE 1 Creating Different Filters	(Continued)
ATTRIBUTES	See attributes(5) for description	ns of the following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO		trvalue_t(3XFN),FN_identifier_t(3XFN), g_t(3XFN),fn_attr_ext_search(3XFN), ributes(5)
		Networking Library Functions 143

## FN\_status\_t(3XFN)

NAME	FN_status_t, fn_status_create, fn_status_destroy, fn_status_copy, fn_status_assign,
	fn_status_code, fn_status_remaining_name, fn_status_resolved_name,
	fn_status_resolved_ref, fn_status_diagnostic_message, fn_status_link_code,
	fn_status_link_remaining_name, fn_status_link_resolved_name,
	fn_status_link_resolved_ref, fn_status_link_diagnostic_message, fn_status_is_success,
	fn_status_set_success, fn_status_set, fn_status_set_code,
	fn_status_set_remaining_name, fn_status_set_resolved_name,
	fn_status_set_resolved_ref, fn_status_set_diagnostic_message,
	fn_status_set_link_code, fn_status_set_link_remaining_name,
	fn_status_set_link_resolved_name, fn_status_set_link_resolved_ref,
	fn_status_set_link_diagnostic_message, fn_status_append_resolved_name,
	fn_status_append_remaining_name, fn_status_advance_by_name, fn_status_description – an XFN status object
SYNOPSIS	cc [flag ] filelxfn [ library ]
	<pre>#include <xfn xfn.h=""></xfn></pre>
	<pre>FN_status_t *fn_status_create(void);</pre>
	<pre>void fn_status_destroy(FN_status_t *stat);</pre>
	<pre>FN_status_t *fn_status_copy(const FN_status_t *stat);</pre>
	<pre>FN_status_t *fn_status_assign(FN_status_t *dst, const FN_status_t</pre>
	unsigned int <b>fn_status_code</b> (const FN_status_t * <i>stat</i> );
	<pre>const FN_composite_name_t  *fn_status_remaining_name(constFN_status_t *stat);</pre>
	<pre>const FN_composite_name_t     *fn status resolved name(constFN status t *stat);</pre>
	<pre>const FN_ref_t *fn_status_resolved_ref(constFN_status_t *stat);</pre>
	<pre>const FN_string_t *fn_status_diagnostic_message(constFN_status_t     *stat);</pre>
	<pre>unsigned int fn_status_link_code(const FN_status_t *stat);</pre>
	<pre>const FN_composite_name_t   *fn_status_link_remaining_name(constFN_status_t *stat);</pre>
	<pre>const FN_composite_name_t     *fn_status_link_resolved_name(constFN_status_t *stat);</pre>
	<pre>const FN_ref_t *fn_status_link_resolved_ref(constFN_status_t     *stat);</pre>
	<pre>const FN_string_t     *fn_status_link_diagnostic_message(constFN_status_t *stat);</pre>
	<pre>int fn_status_is_success(const FN_status_t *stat);</pre>

int **fn status set success**(FN status t \**stat*);

- int **fn status set code** (FN status t \**stat*, unsigned int *code*);
- int fn\_status\_set\_remaining\_name(FN\_status\_t \*stat, const FN composite name t \*name);
- int fn\_status\_set\_resolved\_name(FN\_status\_t \*stat, const
   FN composite name t \*name);
- int fn\_status\_set\_diagnostic\_message(FN\_status\_t \*stat, const FN\_string\_t \*msg);
- int **fn status set link code** (FN status t \**stat*, unsigned int *code*);
- int fn\_status\_set\_link\_remaining\_name(FN\_status\_t \*stat, const FN composite name t \*name);
- int fn\_status\_set\_link\_resolved\_name(FN\_status\_t \*stat, const FN\_composite\_name\_t \*name);
- int fn\_status\_set\_link\_resolved\_ref(FN\_status\_t \*stat, const
   FN ref t \*ref);
- int fn\_status\_set\_link\_diagnostic\_message(FN\_status\_t \*stat, const
   FN\_string\_t \*msg);
- int fn\_status\_append\_resolved\_name(FN\_status\_t \*stat, const FN composite name t \*name);
- int fn\_status\_append\_remaining\_name(FN\_status\_t \*stat, const FN\_composite\_name\_t \*name);
- int fn\_status\_advance\_by\_name(FN\_status\_t \*stat, const
   FN composite name t \*prefix, const FN ref t \*resolved\_ref);

**DESCRIPTION** The result status of operations in the context interface and the attribute interface is encapsulated in an FN\_status\_t object. This object contains information about how the operation completed: whether an error occurred in performing the operation, the nature of the error, and information that helps locate where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.

The context status object consists of several items of information:

### FN\_status\_t(3XFN)

primary status code	An unsigned int code describing the disposition of the operation.
resolved name	In the case of a failure during the resolution phase of the operation, this is the leading portion of the name that was resolved successfully. Resolution may have been successful beyond this point, but the error might not be pinpointed further.
resolved reference	The reference to which resolution was successful (in other words, the reference to which the resolved name is bound).
remaining name	The remaining unresolved portion of the name.
diagnostic message	This contains any diagnostic message returned by the context implementation. This message provides the context implementation a way of notifying the end-user or administrator of any implementation-specific information related to the returned error status. The diagnostic message could then be used by the end-user or administrator to take appropriate out-of-band action to rectify the problem.
link status code	In the case that an error occurred while resolving an XFN link, the primary status code has the value FN_E_LINK_ERROR and the link status code describes the error that occurred while resolving the XFN link.
resolved link name	In the case of a link error, this contains the resolved portion of the name in the XFN link.
resolved link reference	In the case of a link error, this contains the reference to which the resolved link name is bound.
remaining link name	In the case of a link error, this contains the remaining unresolved portion of the name in the XFN link.
link diagnostic message	In the case of a link error, this contains any diagnostic message related to the resolution of the link.
Both the primary status code and the link status code are values of type unsigned int that are drawn from the same set of meaningful values. XFN reserves the values of through 127 for standard meanings. The values and interpretations for the codes are determined by XFN. See xfn_status_codes(3XFN).	
fn_status_create() creates a status object with status FN_SUCCESS. fn_status_destroy() releases the storage associated with <i>stat</i> .	

fn\_status\_destroy() releases the storage associated with stat. fn\_status\_copy() returns a copy of the status object stat. fn\_status\_assign() makes a copy of the status object src and assigns it to dst, releasing any old contents of dst. A pointer to the same object as dst is returned.

	<pre>fn_status_code() returns the status code. fn_status_remaining_name() returns the remaining part of name to be resolved. fn_status_resolved_name() returns the part of the composite name that has been resolved. fn_status_resolved_ref() returns the reference to which resolution was successful. fn_status_diagnostic_message returns any diagnostic message set by the context implementation. fn status link code() returns the link status code.</pre>
	<pre>fn_status_link_remaining_name() returns the remaining part of the link name that has not been resolved. fn_status_link_resolved_name() returns the part of the link name that has been resolved. fn_status_link_resolved_ref() returns the reference to which resolution of the link was successful. fn_status_link_diagnostic_message() returns any diagnostic message set by the context implementation during resolution of the link.</pre>
	<pre>fn_status_is_success() returns 1 if the status indicates success, 0 otherwise.</pre>
	<pre>fn_status_set_success() sets the status code to FN_SUCCESS and clears all other parts of stat. fn_status_set() sets the non-link contents of the status object stat. fn_status_set_code() sets the primary status code field of the status object stat. fn_status_set_remaining_name() sets the remaining name part of the status object stat to name. fn_status_set_resolved_name() sets the resolved name part of the status object stat to name. fn_status_set_resolved_ref() sets the resolved reference part of the status objectstat to ref. fn_status_set_diagnostic_message() sets the diagnostic message part of the status object to msg.</pre>
	<pre>fn_status_set_link_code() sets the link status code field of the status object stat to indicate why resolution of the link failed. fn_status_set_link_remaining_name() sets the remaining link name part of the status object stat to name. fn_status_set_link_resolved_name() sets the resolved link name part of the status object stat to name. fn_status_set_link_resolved_ref() sets the resolved link reference part of the status object stat to ref. fn_status_set_link_diagnostic_message() sets the link diagnostic message part of the status object to msg.</pre>
	<pre>fn_status_append_resolved_name() appends as additional components name to the resolved name part of the status object stat. fn_status_append_remaining_name() appends as additional components name to the remaining name part of the status object stat. fn_status_advance_by_name() removes prefix from the remaining name, and appends it to the resolved name. The resolved reference part is set to resolved_ref. This operation returns 1 on success, 0 if the prefix is not a prefix of the remaining name.</pre>
<b>RETURN VALUES</b>	The fn_status_set_*( ) operations return 1 if the operation succeeds, 0 if the operation fails.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

# FN\_status\_t(3XFN)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	<pre>FN_composite_name_t(3XFN), FN_ref_ xfn(3XFN), xfn_status_codes(3XFN), a</pre>	
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary pecification. It is likely that there will be minor changes to these interfaces to reflect hanges in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As he interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.	

NAME	FN_string_t, fn_string_create, fn_string_destroy, fn_string_from_str, fn_string_from_str_n, fn_string_str, fn_string_from_contents, fn_string_code_set, fn_string_charcount, fn_string_bytecount, fn_string_contents, fn_string_copy, fn_string_assign, fn_string_from_strings, fn_string_from_substring, fn_string_is_empty, fn_string_compare, fn_string_compare_substring, fn_string_next_substring, fn_string_prev_substring – a character string
SYNOPSIS	<pre>cc [ flag ] filelxfn [ library ] #include <xfn xfn.h=""></xfn></pre>
	<pre>FN_string_t *fn_string_create(void);</pre>
	<pre>void fn_string_destroy(FN_string_t *str);</pre>
	<pre>FN_string_t *fn_string_from_str(const unsigned char *cstr);</pre>
	<pre>FN_string_t *fn_string_from_str_n(const unsigned char *cstr,</pre>
	<pre>const unsigned char *fn_string_str(const FN_string_t *str, unsigned int *status);</pre>
	<pre>FN_string_t *fn_string_from_contents (unsigned long code_set, const void *locale_info, size_t locale_info_len, size_t charcount, size_t bytecount, const void *contents, unsigned int *status);</pre>
	<pre>unsigned long fn_string_code_set(const FN_string_t *str, const void **locale_info, size_t *locale_info_len);</pre>
	<pre>size_t fn_string_charcount(const FN_string_t *str);</pre>
	<pre>size_t fn_string_bytecount(const FN_string_t *str);</pre>
	<pre>const void *fn_string_contents(const FN_string_t *str);</pre>
	<pre>FN_string_t *fn_string_copy(const FN_string_t *str);</pre>
	<pre>FN_string_t *fn_string_assign(FN_string_t *dst, const FN_string_t</pre>
	<pre>FN_string_t *fn_string_from_strings(unsigned int *status, const FN_string_t *s1, const FN_string_t *s2,);</pre>
	<pre>FN_string_t *fn_string_from_substring(constFN_string_t *str, int</pre>
	<pre>int fn_string_is_empty(const FN_string_t *str);</pre>
	<pre>int fn_string_compare(const FN_string_t *str1, const FN_string_t     *str2, unsigned int string_case, unsigned int *status);</pre>
	<pre>int fn_string_compare_substring(const FN_string_t *str1, int first,</pre>

Networking Library Functions 149

### FN\_string\_t(3XFN)

- int fn\_string\_next\_substring(const FN\_string\_t \*str, const
   FN\_string\_t \*sub, int index, unsigned int string\_case, unsigned
   int \*status);
- int fn\_string\_prev\_substring(const FN\_string\_t \*str, const
   FN\_string\_t \*sub, int index, unsigned int string\_case, unsigned
   int \*status);
- **DESCRIPTION** The FN\_string\_t type is used to represent character strings in the XFN interface. It provides insulation from specific string representations.

The FN\_string\_t supports multiple code sets. It provides creation functions for character strings of the code set of the current locale setting and a generic creation function for arbitrary code sets. The degree of support for the functions that manipulate FN\_string\_t for arbitrary code sets is implementation-dependent. An XFN implementation is required to support the ISO 646 code set; all other code sets are optional.

fn\_string\_destroy() releases the storage associated with the given string.

fn\_string\_create() creates an empty string.

fn\_string\_from\_str() creates an FN\_string\_t object from the given null terminated string based on the code set of the current locale setting. The number of characters in the string is determined by the code set of the current locale setting. fn\_string\_from\_str\_n() is like fn\_string\_from\_str() except only n characters from the given string are used. fn\_string\_str() returns the contents of the given string str in the form of a null terminated string in the code set and current locale setting.

fn\_string\_from\_contents() creates an FN\_string\_t object using the specified code set *code\_set*, locale information *locale\_info*, and data in the given buffer *contents*. *bytecount* specifies the number of bytes in *contents* and *charcount* specifies the number of characters represented by *contents*.

fn\_string\_code\_set() returns the code set associated with the given string object
and, if present, the locale information in *locale\_info*. fn\_string\_charcount()
returns the number of characters in the given string object.

fn\_string\_bytecount() returns the number of bytes used to represent the given
string object. fn\_string\_contents() returns a pointer to the contents of the given
string object.

fn\_string\_copy() returns a copy of the given string object. fn\_string\_assign() makes a copy of the string object *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned. fn\_string\_from\_strings() is a function that takes a variable number of arguments (minimum of 2), the last of which must be NULL (0); it returns a new string object composed of the left to right concatenation of the given strings, in the given order. The support for strings with different code sets and/or locales as arguments to a single invocation of fn\_string\_from\_strings() is implementation-dependent. fn\_string\_from\_substring() returns a new string object consisting of the characters located between *first* and last inclusive from *str*. Indexing begins with 0. If last is FN\_STRING\_INDEX\_LAST or exceeds the length of the string, the index of the last character of the string is used.

fn string is empty() returns whether *str* is an empty string.

Comparison of two strings must take into account code set and locale information. If strings are in the same code set and same locale, case sensitivity is applied according to the case sensitivity rules applicable for the code set and locale; case sensitivity may not necessarily be relevant for all string encodings. If *string\_case* is non-zero, case is significant and equality for strings of the same code set is defined as equality between byte-wise encoded values of the strings. If *string\_case* is zero, case is ignored and equality for strings of the same code set. Support for comparison between strings of different code sets, or lack thereof, is implementation-dependent.

fn\_string\_compare() compares strings str1 and str2 and returns 0 if they are equal, non-zero if they are not equal. If two strings are not equal, fn\_string\_compare() returns a positive value if the difference of str2 precedes that of str1 in terms of byte-wise encoded value (with case-sensitivity taken into account when string\_case is non-zero), and a negative value if the difference of str1 precedes that of str2, in terms of byte-wise encoded value (with case-sensitivity taken into account when string\_case is non-zero). Such information (positive versus negative return value) may be used by applications that use strings of code sets in which ordering is meaningful; this information is not of general use in internationalized environments. fn\_string\_compare\_substring() is similar to fn\_string\_compare() except that fn\_string\_compare\_substring() compares characters between first and last inclusive of str2 with str1. Comparison of strings with incompatible code sets returns a negative or positive value (never 0) depending on the implementation.

fn\_string\_next\_substring() returns the index of the next occurrence of sub at or after index in the string str. FN\_STRING\_INDEX\_NONE is returned if sub does not occur. fn\_string\_prev\_substring() returns the index of the previous occurrence of sub at or before index in the string str. FN\_STRING\_INDEX\_NONE is returned if sub does not occur. In both of these functions, string\_case specifies whether the search should take case-sensitivity into account.

#### ERRORS

**5** fn\_string\_str() returns 0 and sets *status* to FN\_E\_INCOMPATIBLE\_CODE\_SETS if the given string's representation cannot be converted into the code set of the current locale setting. It is implementation-dependent which code sets can be converted into the code set of the current locale.

Code set mismatches that occur during concatenation, searches, or comparisons are resolved in an implementation-dependent way. When an implementation discovers that arguments to substring searches and comparison operations have incompatible

### FN\_string\_t(3XFN)

code sets, it sets *status* to FN\_E\_INCOMPATIBLE\_CODE\_SETS. In such cases, fn\_string\_from\_strings() returns 0. The returned value for comparison operations when there is code set or locale incompatibility is either negative or positive (greater than 0); it is never 0.

fn\_string\_from\_contents() returns 0 and status is set to
FN\_E\_INCOMPATIBLE\_CODE\_SETS if the supplied code set and/or locale
information are not supported by the XFN implementation.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** xfn(3XFN), attributes(5)

**NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME	getaddrinfo, getnameinfo, freeaddrinfo, gai_strerror – translate between node name and address	
SYNOPSIS	<pre>cc [flag] filelsocket -lnsl [library] #include <sys socket.h=""> #include <netdb.h></netdb.h></sys></pre>	
	<pre>int getaddrinfo(const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res); int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags);</pre>	
	<pre>void freeaddrinfo(struct addrinfo *ai);</pre>	
	<pre>char *gai_strerror(int errcode);</pre>	
<b>DESCRIPTION</b> These functions perform translations from node name to address and from add node name in a protocol-independent manner.		
	The getaddrinfo() function performs the node name to address translation. The <i>nodename</i> and <i>servname</i> arguments are pointers to null-terminated strings or NULL. One or both of these arguments must be a non-null pointer. In the normal client scenario, both the <i>nodename</i> and <i>servname</i> are specified. In the normal server scenario, only the <i>servname</i> is specified. A non-null <i>nodename</i> string can be either a node name or a numeric host address string (a dotted-decimal IPv4 address or an IPv6 hex address). A non-null <i>servname</i> string can be either a service name or a decimal port number. The caller can optionally pass an addrinfo structure, pointed to by the third argument, to provide hints concerning the type of socket that the caller supports.	
	The addrinfo structure is defined as:	
	<pre>struct addrinfo { int ai_flags; /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */ int ai_family; /* PF_xxx */ int ai_socktype; /* SOCK_xxx */ int ai_protocol; /* 0 or IPPROTO_xxx for IPv4 and IPv6 */ size_t ai_addrlen; /* length of ai_addr */ char *ai_canonname; /* canonical name for nodename */ struct sockaddr *ai_addr; /* binary address */ struct addrinfo *ai_next; /* next structure in linked list */ };</pre>	
	In this <i>hints</i> structure, all members other than ai_flags, ai_family, ai_sockty, and ai_protocol must be 0 or a null pointer. A value of PF_UNSPEC for ai_fami indicates that the caller will accept any protocol family. A value of 0 for ai_sockty indicates that the caller will accept any socket type. A value of 0 for ai_protocol indicates that the caller will accept any protocol. For example, if the caller handles only TCP and not UDP, then the ai_socktype member of the <i>hints</i> structure shoul be set to SOCK_STREAM when getaddrinfo() is called. If the caller handles only IPv4 and not IPv6, then the ai_family member of the <i>hints</i> structure should be set PF_INET when getaddrinfo() is called. If the third argument to getaddrinfo()	

### getaddrinfo(3SOCKET)

is a null pointer, it is as if the caller had filled in an addrinfo structure initialized to 0 with ai\_family set to PF\_UNSPEC.

Upon success, a pointer to a linked list of one or more addrinfo structures is returned through the final argument. The caller can process each addrinfo structure in this list by following the ai\_next pointer, until a null pointer is encountered. In each returned addrinfo structure the three members ai\_family, ai\_socktype, and ai\_protocol are the corresponding arguments for a call to the socket(3SOCKET) function. In each addrinfo structure the ai\_addr member points to a filled-in socket address structure whose length is specified by the ai addrlen member.

If the AI\_PASSIVE bit is set in the ai\_flags member of the *hints* structure, the caller plans to use the returned socket address structure in a call to bind(3SOCKET). In this case, if the *nodename* argument is a null pointer, the IP address portion of the socket address structure will be set to INADDR\_ANY for an IPv4 address or IN6ADDR ANY INIT for an IPv6 address.

If the AI\_PASSIVE bit is not set in the ai\_flags member of the *hints* structure, then the returned socket address structure will be ready for a call to connect(3SOCKET) (for a connection-oriented protocol) or either connect(3SOCKET), sendto(3SOCKET), or sendmsg(3SOCKET) (for a connectionless protocol). If the *nodename* argument is a null pointer, the IP address portion of the socket address structure will be set to the loopback address.

If the AI\_CANONNAME bit is set in the ai\_flags member of the *hints* structure, then upon successful return the ai\_canonname member of the first addrinfo structure in the linked list will point to a null-terminated string containing the canonical name of the specified *nodename*.

If the AI\_NUMERICHOST bit is set in the ai\_flags member of the *hints* structure, then a non-null *nodename* string must be a numeric host address string. Otherwise an error of EAI\_NONAME is returned. This flag prevents any type of name resolution service (such as DNS) from being called.

All of the information returned by getaddrinfo() is dynamically allocated: the addrinfo structures as well as the socket address structures and canonical node name strings pointed to by the addrinfo structures. The freeaddrinfo() function is called to return this information to the system the function. For freeaddrinfo(), the addrinfo structure pointed to by the *ai* argument is freed, along with any dynamic storage pointed to by the structure. This operation is repeated until a null ai next pointer is encountered.

To aid applications in printing error messages based on the EAI\_\* codes returned by getaddrinfo(), the gai\_strerror() is defined. The argument is one of the EAI\_\* values defined below and the return value points to a string describing the error. If the argument is not one of the EAI\_\* values, the function still returns a pointer to a string whose contents indicate an unknown error.

### getaddrinfo(3SOCKET)

The getnameinfo() function looks up an IP address and port number provided by the caller in the name service database and system-specific database, and returns text strings for both in buffers provided by the caller. The function indicates successful completion by a 0 return value; a non-zero return value indicates failure.

The first argument, *sa*, points to either a sockaddr\_in structure (for IPv4) or a sockaddr\_in6 structure (for IPv6) that holds the IP address and port number. The *salen* argument gives the length of the sockaddr\_in or sockaddr\_in6 structure.

The function returns the node name associated with the IP address in the buffer pointed to by the *host* argument. The caller provides the size of this buffer with the *hostlen* argument. The service name associated with the port number is returned in the buffer pointed to by *serv*, and the *servlen* argument gives the length of this buffer. The caller specifies not to return either string by providing a 0 value for the *hostlen* or *servlen* arguments. Otherwise, the caller must provide buffers large enough to hold the node name and the service name, including the terminating null characters.

To aid the application in allocating buffers for these two returned strings, the following constants are defined in <netdb.h>:

#define NI\_MAXHOST 1025
#define NI\_MAXSERV 32

The final argument is a flag that changes the default actions of this function. By default, the fully-qualified domain name (FQDN) for the host is looked up in the name service database and returned. If the flag bit NI\_NOFQDN is set, only the node name portion of the FQDN is returned for local hosts.

If the flag bit NI\_NUMERICHOST is set, or if the host's name cannot be located in the name service, the numeric form of the host's address is returned instead of its name, for example, by calling inet\_ntop() (see inet(3SOCKET)) instead of getipnodebyname(3SOCKET). If the flag bit NI\_NAMEREQD is set, an error is returned if the host's name cannot be located in the name service database.

If the flag bit NI\_NUMERICSERV is set, the numeric form of the service address is returned (for example, its port number) instead of its name. The two NI\_NUMERIC\* flags are required to support the "-n" flag that many commands provide.

A fifth flag bit, NI\_DGRAM, specifies that the service is a datagram service, and causes getservbyport(3SOCKET) to be called with a second argument of "udp" instead of the default "tcp". This is required for the few ports (for example, 512-514) that have different services for UDP and TCP.

These NI\_\* flags are defined in <netdb.h> along with the AI\_\* flags already defined for getaddrinfo().

**RETURN VALUES** For getaddrinfo(), if the query is successful, a pointer to a linked list of one or more addrinfo structures is returned by the fourth argument and the function returns 0. If the query fails, a non-zero error code will be returned. For getnameinfo(), if successful, the strings hostname and service are copied into *host* 

### getaddrinfo(3SOCKET)

and *serv*, respectively. If unsuccessful, zero values for either *hostlen* or *servlen* will suppress the associated lookup; in this case no data is copied into the applicable buffer. If gai\_strerror() is successful, a pointer to a string containing an error message appropriate for the EAI\_\* errors is returned. If *errcode* is not one of the EAI\_\* values, a pointer to a string indicating an unknown error is returned.

**ERRORS** The following names are the error values returned by getaddrinfo() and are defined in <netdb.h>:

EAI_ADDRFAMILY address family for nodename not supported EAI_AGAIN temporary failure in name resolution EAI_BADFLAGS invalid value for ai_flags EAI_FAIL non-recoverable failure in name resolution EAI_FAMILY ai_family not supported EAI_MEMORY memory allocation failure EAI_NODATA no address associated with nodename EAI_NONAME nodename nor servname provided, or not known EAI_SERVICE servname not supported for ai_socktype EAI_SOCKTYPE ai_socktype not supported EAI_SYSTEM system error returned in errno	
/etc/inet/hosts	
/etc/inet/ipnodes	
/etc/netconfig	
/etc/nsswitch.conf	
<pre>/etc/nsswitch.conf gethostbyname(3NSL), getipnodebyname(3SOCKET), htonl(3SOCKET), inet(3SOCKET), netdb(3HEAD), socket(3SOCKET), hosts(4), ipnodes(4), nsswitch.conf(4)</pre>	

gethostbyname, gethostbyname r, gethostbyaddr, gethostbyaddr r, gethostent, NAME gethostent\_r, sethostent, endhostent - get network host entry SYNOPSIS cc [ flag ... ] file ... - lnsl [ library ... ] #include <netdb.h> struct hostent \*gethostbyname(const char \*name); struct hostent **\*gethostbyname r**(const char **\****name*, struct hostent \*result, char \*buffer, intbuflen, int \*h\_errnop); struct hostent \*gethostbyaddr(const char \*addr, int len, int type); struct hostent \*gethostbyaddr r(const char \*addr, int length, int type, struct hostent \*result, char \*buffer, int buflen, int \*h\_errnop); struct hostent \*gethostent(void); struct hostent \*gethostent r(struct hostent \*result, char \*buffer, int buflen, int \*h\_errnop); int sethostent(int stayopen); int endhostent(void); DESCRIPTION These functions are used to obtain entries describing hosts. An entry may come from any of the sources for hosts specified in the /etc/nsswitch.conf file. See nsswitch.conf(4). Please take note that these functions have been superseded by the newer functions, getipnodebyname(3SOCKET), getipnodebyaddr(3SOCKET), and getaddrinfo(3SOCKET). The newer functions provide greater portability to applications when multithreading is done or technologies such as IPv6 are used. For example, the functions described below cannot be used with applications targeted to work with IPv6. gethostbyname() searches for information for a host with the hostname specified by the character-string parameter name. gethostbyaddr() searches for information for a host with a given host address. The parameter type specifies the family of the address. This should be one of the address families defined in <sys/socket.h>. The parameter *addr* must be a pointer to a buffer containing the address. The address is given in a form specific to the address family. See the NOTES section below for more information. Also see the EXAMPLES section below on how to convert a "." separated Internet IP address notation into the *addr* parameter. The parameter *len* specifies the length of the buffer indicated by *addr*. All addresses are returned in network order. In order to interpret the addresses, byteorder(3SOCKET) must be used for byte order conversion. The functions sethostent(), gethostent(), and endhostent() are used to enumerate host entries from the database.

genosibynanie(SNSL)		
	<pre>sethostent() sets (or resets) the enumeration to the beginning of the set of host entries. This function should be called before the first call to gethostent(). Calls to gethostbyname() and gethostbyaddr() leave the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endhostent().</pre>	
	Successive calls to gethostent() return either successive entries or NULL, indicating the end of the enumeration.	
	<pre>endhostent() may be called to indicate that the caller expects to do no further host entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more host retrieval functions after calling endhostent().</pre>	
Reentrant Interfaces	The functions gethostbyname(), gethostbyaddr(), and gethostent() use static storage that is reused in each call, making these functions unsafe for use in multi-threaded applications.	
	The functions gethostbyname_r(), gethostbyaddr_r(), and gethostent_r() provide reentrant interfaces for these operations.	
	Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multi-threaded applications.	
	Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter <i>result</i> must be a pointer to a struct hostent structure allocated by the caller. On successful completion, the function returns the host entry in this structure. The parameter <i>buffer</i> must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the host data. All of the pointers within the returned struct hostent <i>result</i> point to data stored within this buffer. See RETURN VALUES. The buffer must be large enough to hold all of the data associated with the host entry. The parameter <i>buflen</i> should give the size in bytes of the buffer indicated by <i>buffer</i> . The parameter <i>h_errnop</i> should be a pointer to an integer. An integer error status value is stored there on certain error conditions. See ERRORS.	
	For enumeration in multi-threaded applications, the position within the enumeration is a process-wide property shared by all threads. <pre>sethostent()</pre> may be used in a multi-threaded application but resets the enumeration position for all threads. If multiple threads interleave calls to <pre>gethostent_r()</pre> , the threads will enumerate disjoint subsets of the host database.	
	Like their non-reentrant counterparts, gethostbyname_r() and gethostbyaddr_r() leave the enumeration position in an indeterminate state.	
RETURN VALUES	Host entries are represented by the struct hostent structure defined in <netdb.h>:</netdb.h>	

158 man pages section 3: Networking Library Functions • Last Revised 17 Nov 1999

```
struct hostent {
               char *h_name; /* canonical name of host */
char **h_aliases; /* alias list */
int h_addrtype; /* host address type */
int h_length; /* length of address */
               char *h_name;
                char **h_addr_list; /* list of addresses */
           };
           See the EXAMPLES section below for information about how to retrieve a "." separated
           Internet IP address string from the h addr list field of struct hostent.
           The functions gethostbyname(), gethostbyname r(), gethostbyaddr(), and
           gethostbyaddr r() each return a pointer to a struct hostent if they
           successfully locate the requested entry; otherwise they return NULL.
           The functions gethostent() and gethostent r() each return a pointer to a
           struct hostent if they successfully enumerate an entry; otherwise they return
           NULL, indicating the end of the enumeration.
           The functions gethostbyname(), gethostbyaddr(), and gethostent() use
           static storage, so returned data must be copied before a subsequent call to any of these
           functions if the data is to be saved.
           When the pointer returned by the reentrant functions gethostbyname r(),
           gethostbyaddr r(), and gethostent r() is not NULL, it is always equal to the
           result pointer that was supplied by the caller.
           The functions sethostent() and endhostent() return 0 on success.
ERRORS
           The reentrant functions gethostbyname r(), gethostbyaddr r(), and
           gethostent r() will return NULL and set errno to ERANGE if the length of the buffer
           supplied by caller is not large enough to store the result. See Intro(2) for the proper
           usage and interpretation of errno in multithreaded applications.
            The reentrant functions gethostbyname r() and gethostbyaddr r() set the
           integer pointed to by h_errnop to one of these values in case of error.
           On failures, the non-reentrant functions gethostbyname() and gethostbyaddr()
           set a global integer h_errno to indicate one of these error codes (defined in
            <netdb.h>): HOST NOT FOUND, TRY AGAIN, NO RECOVERY, NO DATA, and
           NO ADDRESS.
           Note however that if a resolver is provided with a malformed address, or if any other
           error occurs before gethostbyname() is resolved, then gethostbyname() returns
           an internal error with a value of -1.
           gethostbyname() will set h_errno to NETDB INTERNAL when it returns a NULL
           value.
```

**EXAMPLES** | **EXAMPLE 1** Using gethostbyname()

Here is a sample program that gets the canonical name, aliases, and "." separated Internet IP addresses for a given "." separated IP address: #include <stdio.h> #include <string.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <netdb.h> main(int argc, const char \*\*argv) { ulong\_t addr; struct hostent \*hp; char \*\*p; if (argc != 2) { (void) printf("usage: %s IP-address\n", argv[0]); exit (1); if ((int)(addr = inet\_addr(argv[1])) == -1) { (void) printf("IP-address must be of the form a.b.c.d\n"); exit (2); } hp = gethostbyaddr((char \*)&addr, sizeof (addr), AF INET); if (hp == NULL) { (void) printf("host information for %s not found\n", argv[1]); exit (3); for (p = hp->h addr list; \*p != 0; p++) { struct in\_addr in; char \*\*q; (void) memcpy(&in.s addr, \*p, sizeof (in.s addr)); (void) printf("%s\t%s", inet\_ntoa(in), hp->h\_name); for  $(q = hp - >h_aliases; *q != 0; q++)$ (void) printf(" %s", \*q); (void) putchar( $' \ );$ } exit (0); }

Note that the above sample program is unsafe for use in multithreadeded applications.

### FILES | /etc/hosts

/etc/netconfig

/etc/nsswitch.conf

### ATTRIBUTES

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

ATTRIBUTE TYPE ATTRIBUTE VALUE
See "Reentrant Interfaces" in DESCRIPTION.
See "Reentrant Interfaces" in DE

# SEE ALSO Intro(2), Intro(3), byteorder(3SOCKET), inet(3SOCKET), netdir(3NSL), hosts(4), netconfig(4), nsswitch.conf(4), attributes(5), netdb(3HEAD)

# WARNINGS The reentrant interfaces gethostbyname\_r(), gethostbyaddr\_r(), and gethostent\_r() are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

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

In order to ensure that they all return consistent results, gethostbyname(), gethostbyname\_r(), and netdir\_getbyname() are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the inet family entries in netconfig(4) and the hosts: entry in nsswitch.conf(4). Similarly, gethostbyaddr(), gethostbyaddr\_r(), and netdir\_getbyaddr() are implemented in terms of the same internal library function. If the inet family entries in netconfig(4) have a "-" in the last column for nametoaddr libraries, then the entry for hosts in nsswitch.conf will be used; otherwise the nametoaddr libraries in that column will be used, and nsswitch.conf will not be consulted.

There is no analogue of gethostent() and gethostent\_r() in the netdir functions, so these enumeration functions go straight to the hosts entry in nsswitch.conf. Thus enumeration may return results from a different source than that used by gethostbyname(), gethostbyname\_r(), gethostbyaddr(), and gethostbyaddr\_r().

All the functions that return a struct hostent must always return the *canonical name* in the *h\_name* field. This name, by definition, is the well-known and official hostname shared between all aliases and all addresses. The underlying source that satisfies the request determines the mapping of the input name or address into the set of names and addresses in hostent. Different sources might do that in different ways. If there is more than one alias and more than one address in hostent, no pairing is implied between them.

The system will strive to put the addresses on the same subnet as that of the caller first.

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

Use of the enumeration interfaces gethostent() and gethostent\_r() is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in nswitch.conf(4).

The current implementations of these functions only return or accept addresses for the Internet address family (type AF\_INET).

The form for an address of type AF\_INET is a struct in\_addr defined in <netinet/in.h>. The functions described in inet(3SOCKET), and illustrated in the EXAMPLES section above, are helpful in constructing and manipulating addresses in this form.

		gethosthame(overlar)
NAME	gethostname – get name of current host	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <unistd.h></unistd.h></pre>	
	<pre>int gethostname(char *name, size_</pre>	t namelen);
DESCRIPTION	The gethostname() function returns the standard host name for the current machine. The <i>namelen</i> argument specifies the size of the array pointed to by the <i>name</i> argument. The returned name is null-terminated, except that if <i>namelen</i> is an insufficient length to hold the host name, then the returned name is truncated and it is unspecified whether the returned name is null-terminated.	
	Host names are limited to 255 bytes.	
<b>RETURN VALUES</b>	On successful completion, 0 is returned. Ot	herwise, –1 is returned.
ERRORS	No errors are defined.	
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	ATTRIBUTE TYPE MT-Level	ATTRIBUTE VALUE MT-Safe
SEE ALSO		MT-Safe
SEE ALSO	MT-Level	MT-Safe

# getipnodebyname(3SOCKET)

nphodedyname(3SOCKET)			
NAME	getipnodebyname, getipnodebyaddr, freehostent – get IP node entry		
SYNOPSIS	cc [flag] filelsocket -lnsl [library]		
	<pre>#include <sys socket.h=""> #include <netdb.h></netdb.h></sys></pre>		
	<pre>struct hostent *getipnodebyname(const char *name, int af, int flags, int *error_num);</pre>		
	<pre>struct hostent *getipnodebyaddr(const void *src, size_t len, int af, int *error_num);</pre>		
	<pre>void freehostent(struct hostent *ptr);</pre>		
DESCRIPTION	The getipnodebyname() function searches the ipnodes database from the beginning and finds the first entry for which the hostname specified by name matches the h_name member. It takes an <i>af</i> argument that specifies the address family, which can be either AF_INET for IPv4 addresses or AF_INET6 for IPv6 addresses. The <i>flags</i> argument determines what results will be returned based on the value of <i>flags</i> . If the <i>flags</i> argument is set to 0 (zero), then the default operation of this function is specified as follows:		
	<ul> <li>If the <i>af</i> argument is AF_INET, then a query is made for an IPv4 address. If successful, IPv4 addresses are returned and the h_length member of the hostent structure will be 4. Otherwise, the function returns a null pointer.</li> </ul>		
	<ul> <li>If the <i>af</i> argument is AF_INET6, then a query is made for an IPv6 address. If successful, IPv6 addresses are returned and the h_length member of the hostent structure will be 16. Otherwise, the function returns a null pointer.</li> </ul>		
	The <i>flags</i> argument changes the default actions of the function. You can set the <i>flags</i> argument by logically ORing any of the following values together:		
	AI_V4MAPPED AI_ALL AI ADDRCONFIG		
	Note that a special flags value of AI_DEFAULT, as defined below, should handle most applications. In other words, porting simple applications to use IPv6 replaces the call		
	<pre>hptr = gethostbyname(name);</pre>		
	with		
	<pre>hptr = getipnodebyname(name, AF_INET6, AI_DEFAULT);</pre>		
	A <i>flags</i> of 0 implies a strict interpretation of the <i>af</i> argument:		
	<ul> <li>If flags is 0 and <i>af</i> is AF_INET, then the caller wants only IPv4 addresses. A query is made for A records. If successful, the IPv4 addresses are returned and the h_length member of the hostent structure will be 4; otherwise, the function returns a null pointer.</li> </ul>		
I I I I I I I I I I I I I I I I I I I			

If *flags* is 0, and if *af* is AF\_INET6, then the caller wants only IPv6 addresses. A query is made for AAAA records. If successful, the IPv6 addresses are returned and the h\_length member of the hostent structure will be 16; otherwise, the function returns a null pointer.

Other constants can be logically-ORed into the *flags* argument, to modify the behavior of the function.

- If the AI\_V4MAPPED flag is specified along with an *af* of AF\_INET6, then the caller can accept IPv4-mapped IPv6 addresses. That is, if no AAAA records are found, then a query is made for A records, and any found are returned as IPv4-mapped IPv6 addresses (h\_length is 16). The AI\_V4MAPPED flag is ignored unless *af* equals AF INET6.
- The AI\_ALL flag is used in conjunction with the AI\_V4MAPPED flag, and is only used with the IPv6 address family. When AI\_ALL is logically ORed with AI\_V4MAPPED flag then the caller wants all addresses: IPv6 and IPv4-mapped IPv6. A query is first made for AAAA records and if successful, the IPv6 addresses are returned. Another query is then made for A records, and any found are returned as IPv4-mapped IPv6 addresses. h\_length is 16. Only if both queries fail does the function return a null pointer. This flag is ignored unless *af* equals AF\_INET6.
- The AI\_ADDRCONFIG flag specifies that a query for AAAA records should occur only if the node has at least one IPv6 source address configured. A query for A records should occur only if the node has at least one IPv4 source address configured. For example, if the node has no IPv6 source addresses configured, and *af* equals AF\_INET6, and the node name being looked up has both AAAA and A records, then:
  - 1. If only AI ADDRCONFIG is specified, the function returns a null pointer.
  - 2. If AI\_ADDRCONFIG or AI\_V4MAPPED is specified, the A records are returned as IPv4-mapped IPv6 addresses.

The special flags value of AI\_DEFAULT is defined as

#define AI\_DEFAULT (AI\_V4MAPPED | AI\_ADDRCONFIG)

The getipnodebyname() function must allow the *name* argument to be either a node name or a literal address string, that is, a dotted-decimal IPv4 address or an IPv6 hex address. This saves applications from having to call inet\_pton(3SOCKET) to handle literal address strings.

Four scenarios arise based on the type of literal address string and the value of the *af* argument. The two simple cases are when *name* is a dotted-decimal IPv4 address and *af* equals AF\_INET, or when *name* is an IPv6 hex address and *af* equals AF\_INET6. The members of the returned hostent structure are:

h name Points to a copy of the name argument

Is a null pointer.

h\_aliases

Networking Library Functions 165

getipnodebyname(3SOCKET)

	h_addrtype	Is a copy of the <i>af</i> argument.
	h_length	Is either 4 (for AF_INET) or 16 (for AF_INET6).
	h_addr_list[0]	Is a pointer to the 4-byte or 16-byte binary address.
	h_addr_list[1]	Is a null pointer
PARAMETERS	af	Address family
	flags	Various flags
	name	Name of host
	error_num	Error storage
	STC	Address for lookup
	len	Length of address
	ptr	Pointer to hostent structure
,		getipnodebyname() and getipnodebyaddr() Otherwise they return NULL.
		not change from its existing definition when used with example, host entries are represented by the struct <netdb.h>:</netdb.h>
	int h_addr int h_leng char **h_ad };An error occurs v	e; /* canonical name of host */ iases; /* alias list */ type; /* host address type */ th; /* length of address */ dr_list; /* list of addresses */ vhen <i>name</i> is an IPv6 hex address and <i>af</i> equals rn value is a null pointer and error_num equals
	The getipnodebyaddr() function has the same arguments as the existing gethostbyaddr(3NSL) function, but adds an error number. As with getipnodebyname(), getipnodebyaddr() is thread safe. The error_num value is returned to the caller with the appropriate error code to support thread safe error code returns. The following error conditions can be returned for error_num:	
	HOST_NOT_FOUND	Host is unknown.
	NO_DATA	No address is available for the <i>name</i> specified in the server request. This error is not a soft error. Another type of <i>name</i> server request might be successful.
	NO_RECOVERY	An unexpected server failure occurred, which is a nonrecoverable error.

	TRY_AGAIN	This error is a soft error that indicates that the local server did not receive a response from an authoritative server. A retry at some later time might be successful.		
	One possible source of confusion is the handling of IPv4-mapped IPv6 addresses and IPv4-compatible IPv6 addresses, but the following logic should apply:			
	IPv6 address or an IPv4-co	<i>n</i> equals 16, and if the IPv6 address is an IPv4-mapped mpatible IPv6 address, then skip over the first 12 bytes o AF_INET, and set <i>len</i> to 4.		
	2. If <i>af</i> is AF_INET, lookup the <i>name</i> for the given IPv4 address.			
	3. If <i>af</i> is AF_INET6, lookup the <i>name</i> for the given IPv6 address.			
	4. If the function is returning success, then the single address that is returned in the hostent structure is a copy of the first argument to the function with the same address family that was passed as an argument to this function.			
	All four steps listed are perfor	med in order.		
	This structure, and the information pointed to by this structure, are dynamically allocated by getipnodebyname() and getipnodebyaddr(). The freehostent function frees this memory.			
EXAMPLES	<b>MPLES EXAMPLE 1</b> Getting the Canonical Name, Aliases, and Internet IP Addresses for a Hostname			
	The following is a sample program that retrieves the canonical name, aliases, and all Internet IP addresses, both version 6 and version 4, for a given hostname.			
	<pre>#include <stdio.h> #include <string.h> #include <sys types.h=""> #include <sys socket.h=""> #include <netinet in.h=""> #include <arpa inet.h=""> #include <arpa inet.h=""></arpa></arpa></netinet></sys></sys></string.h></stdio.h></pre>			
	<pre>main(int argc, const ch {     char abuf[INET6_ADDRSTR     int error_num;     struct hostent *hp;     char **p;</pre>			
	if (argc != 2) { (void) printf(" ", argv[0]); exit (1); }	usage: %s hostname\		
		nter to a hostname or literal IP address */ gv[1], AF_INET6, AI_ALL   AI_ADDRCONFIG   num);		

getipnodebyname(3SOCKET)

EXAMPLE 1 Getting the Canonical Name, Aliases, and Internet IP Addresses for a Given Hostname (Continued) if (hp == NULL) { if (error\_num == TRY\_AGAIN) { printf("%s: unknown host or invalid literal address " "(try again later)\n", argv[1]); } else { printf("%s: unknown host or invalid literal address\n", argv[1]); } exit (1); } for (p = hp->h\_addr\_list; \*p != 0; p++) { struct in6\_addr in6; char \*\*q; bcopy(\*p, (caddr\_t)&in6, hp->h\_length); (void) printf("%s\t%s", inet ntop(AF\_INET6, (void \*)&in6, abuf, sizeof(abuf)), hp->h\_name); for (q = hp->h aliases; \*q != 0; q++) (void) printf(" %s", \*q); (void) putchar('\n'); } freehostent(hp); exit (0); } FILES /etc/inet/hosts /etc/inet/ipnodes /etc/netconfig /etc/nsswitch.conf

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

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWcsl, SUNWarc (32-bit)
		SUNWcslx (64-bit)
	MT Level	Safe
SEE ALSO	getaddrinfo(3SOCKET), gethostbyna	ne(3NSL), hton1(3SOCKET),

- inet(3SOCKET), netdb(3HEAD), hosts(4), ipnodes(4), nsswitch.conf(4)NOTESPrograms 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.
- 168 man pages section 3: Networking Library Functions Last Revised 12 Oct 2000

No enumeration functions are provided for IPv6. Existing enumeration functions, for example, sethostent(3NSL) does not work in combination with getipnodebyname() and getipnodebyaddr().

All the functions that return a struct hostent must always return the canonical in the h\_name field. This name, by definition, is the well-known and official hostname shared between all aliases and all addresses. The underlying source that satisfies the request determines the mapping of the input name or address into the set of names and addresses in hostent. Different sources might do that in different ways. If more than one alias and more than one address in hostent exist, no pairing is implied between them.

The current implementations of these functions only return or accept addresses for the Internet address family (type AF\_INET) or the Internet address family Version 6 (type AF\_INET6).

The form for an address of type AF\_INET is a struct in\_addr defined in <netinet/in.h>. The form for an address of type AF\_INET6 is a struct in6\_addr, defined also in <netinet/in.h>. The functions described in inet\_ntop(3SOCKET) and inet\_pton(3SOCKET) that are illustrated in the EXAMPLES section are helpful in constructing and manipulating addresses in either of these forms.

# getnetbyname(3SOCKET)

metoyname(SSOCKE1)		
NAME	getnetbyname, getnetbyname_r, getnetbyaddr, getnetbyaddr_r, getnetent, getnetent_r, setnetent, endnetent – get network entry	
SYNOPSIS	<b>5 cc</b> [ <i>flag</i> ] <i>file</i> lsocket -lnsl [ <i>library</i> ] #include <netdb.h></netdb.h>	
	<pre>struct netent *getnetbyname(const char *name);</pre>	
	<pre>struct netent *getnetbyname_r(const char *name, struct netent</pre>	
	<pre>struct netent *getnetbyaddr(long net, inttype);</pre>	
	<pre>struct netent *getnetbyaddr_r(long net, inttype, struct netent  *result, char *buffer, int buflen);</pre>	
	<pre>struct netent *getnetent(void);</pre>	
	<pre>struct netent *getnetent_r(struct netent *result, char *buffer, int</pre>	
	<pre>int setnetent(int stayopen);</pre>	
	<pre>int endnetent(void);</pre>	
DESCRIPTION	These functions are used to obtain entries for networks. An entry may come from any of the sources for networks specified in the /etc/nsswitch.conf file. See nsswitch.conf(4).	
	getnetbyname() searches for a network entry with the network name specified by the character string parameter <i>name</i> .	
	getnetbyaddr() searches for a network entry with the network address specified by <i>net</i> . The parameter type specifies the family of the address. This should be one of the address families defined in <sys socket.h="">. See the NOTES section below for more information.</sys>	
	All addresses are returned in network order. In order to interpret the addresses, byteorder(3SOCKET) must be used for byte order conversion.	
	The functions setnetent(), getnetent(), and endnetent() are used to enumerate network entries from the database.	
	<pre>setnetent() sets (or resets) the enumeration to the beginning of the set of network entries. This function should be called before the first call to getnetent(). Calls to getnetbyname() and getnetbyaddr() leave the enumeration position in an indeterminate state. If the stayopen flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endnetent().</pre>	
	Successive calls to getnetent() return either successive entries or NULL, indicating the end of the enumeration.	

170 man pages section 3: Networking Library Functions • Last Revised 23 May 1998

# getnetbyname(3SOCKET)

	8,
	<pre>endnetent() may be called to indicate that the caller expects to do no further network entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more network entry retrieval functions after calling endnetent().</pre>
Reentrant Interfaces	The functions getnetbyname(), getnetbyaddr(), and getnetent() use static storage that is reused in each call, making these routines unsafe for use in multi-threaded applications.
	The functions $getnetbyname_r()$ , $getnetbyaddr_r()$ , and $getnetent_r()$ provide reentrant interfaces for these operations.
	Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multi-threaded applications.
	Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter <i>result</i> must be a pointer to a struct netent structure allocated by the caller. On successful completion, the function returns the network entry in this structure. The parameter <i>buffer</i> must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the network entry data. All of the pointers within the returned struct netent <i>result</i> point to data stored within this buffer. See RETURN VALUES. The buffer must be large enough to hold all of the data associated with the network entry. The parameter <i>buffer</i> should give the size in bytes of the buffer indicated by <i>buffer</i> .
	For enumeration in multi-threaded applications, the position within the enumeration is a process-wide property shared by all threads. <pre>setnetent()</pre> may be used in a multi-threaded application but resets the enumeration position for all threads. If multiple threads interleave calls to <pre>getnetent_r()</pre> , the threads will enumerate disjointed subsets of the network database.
	Like their non-reentrant counterparts, getnetbyname_r() and getnetbyaddr_r() leave the enumeration position in an indeterminate state.
RETURN VALUES	Network entries are represented by the struct netent structure defined in <netdb.h>.</netdb.h>
	The functions getnetbyname(), getnetbyname_r(), getnetbyaddr(), and getnetbyaddr_r() each return a pointer to a struct netent if they successfully locate the requested entry; otherwise they return NULL.
	The functions getnetent() and getnetent_r() each return a pointer to a struct netent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

### getnetbyname(3SOCKET)

The functions getnetbyname(), getnetbyaddr(), and getnetent() use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions getnetbyname\_r(), getnetbyaddr\_r(), and getnetent\_r() is non-NULL, it is always equal to the *result* pointer that was supplied by the caller.

The functions setnetent() and endnetent() return 0 on success.

**ERRORS** The reentrant functions getnetbyname\_r(), getnetbyaddr\_r() and getnetent\_r() will return NULL and set *errno* to ERANGE if the length of the buffer supplied by caller is not large enough to store the result. See intro(2) for the proper usage and interpretation of *errno* in multi-threaded applications.

FILES /etc/networks

/etc/nsswitch.conf

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

- SEE ALSO Intro(2), Intro(3), byteorder(3SOCKET), inet(3SOCKET), networks(4), nsswitch.conf (4), attributes(5), netdb(3HEAD)
- **WARNINGS** The reentrant interfaces getnetbyname\_r(), getnetbyaddr\_r(), and getnetent\_r() are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.
  - **NOTES** The current implementation of these functions only return or accept network numbers for the Internet address family (type AF\_INET). The functions described in inet(3SOCKET) may be helpful in constructing and manipulating addresses and network numbers in this form.

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

When compiling multi-threaded applications, see Intro(3), *Notes On Multithread Applications*, for information about the use of the REENTRANT flag.

Use of the enumeration interfaces getnetent() and getnetent\_r() is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in nsswitch.conf(4).

	genieconing(SNSL)	
NAME	getnetconfig, setnetconfig, endnetconfig, getnetconfigent, freenetconfigent, nc_perror, nc_sperror – get network configuration database entry	
SYNOPSIS	<pre>#include <netconfig.h></netconfig.h></pre>	
	<pre>struct netconfig *getnetconfig(void *handlep);</pre>	
	<pre>void *setnetconfig(void);</pre>	
	<pre>int endnetconfig(void *handlep);</pre>	
	<pre>struct netconfig *getnetconfigent(const char *netid);</pre>	
	<pre>void freenetconfigent(struct netconfig *netconfigp);</pre>	
	<pre>void nc_perror(const char *msg);</pre>	
	char <b>*nc_sperror</b> (void);	
DESCRIPTION	The library routines described on this page are part of the Network Selection component. They provide the application access to the system network configuration database, /etc/netconfig. In addition to the routines for accessing the netconfig database, Network Selection includes the environment variable NETPATH (see environ(5)) and the NETPATH access routines described in getnetpath(3NSL).	
	<pre>getnetconfig() returns a pointer to the current entry in the netconfig database, formatted as a struct netconfig. Successive calls will return successive netconfig entries in the netconfig database.getnetconfig() can be used to search the entire netconfig file.getnetconfig() returns NULL at the end of the file. handlep is the handle obtained through setnetconfig().</pre>	
	A call to setnetconfig() has the effect of "binding" to or "rewinding" the netconfig database. setnetconfig() must be called before the first call to getnetconfig() and may be called at any other time. setnetconfig() need not be called before a call to getnetconfigent().setnetconfig() returns a unique handle to be used by getnetconfig().	
	<pre>endnetconfig() should be called when processing is complete to release resources for reuse. handlep is the handle obtained through setnetconfig(). Programmers should be aware, however, that the last call to endnetconfig() frees all memory allocated by getnetconfig() for the struct netconfig data structure. endnetconfig() may not be called before setnetconfig().</pre>	
	getnetconfigent() returns a pointer to the struct netconfig structure corresponding to <i>netid</i> . It returns NULL if <i>netid</i> is invalid (that is, does not name an entry in the netconfig database).	
	<pre>freenetconfigent() frees the netconfig structure pointed to by netconfigp (previously returned by getnetconfigent()).</pre>	

getnetconfig(3NSL)			
	nc_perror() prints a message to the standard error indicating why any of the above routines failed. The message is prepended with the string <i>msg</i> and a colon. A NEWLINE is appended at the end of the message.		
	nc_sperror() is similar to nc_perror() standard error, will return a pointer to a str	8	
	<pre>nc_perror() and nc_sperror() can als routines defined in getnetpath(3NSL).</pre>	so be used with the NETPATH access	
RETURN VALUES	<pre>setnetconfig() returns a unique handle case of an error, setnetconfig() returns nc_sperror() can be used to print the returns</pre>	NULL and nc_perror() or	
	getnetconfig() returns a pointer to the database, formatted as a struct netconf the end of the file, or upon failure.	-	
	endnetconfig() returns 0 on success and -1 on failure (for example, if setnetconfig() was not called previously).		
	On success, getnetconfigent() returns a pointer to the struct netconfig structure corresponding to <i>netid</i> ; otherwise it returns NULL.		
	nc_sperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.		
ATTRIBUTES	See attributes (5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	<pre>getnetpath(3NSL), netconfig(4), attributes(5), environ(5)</pre>		

174 man pages section 3: Networking Library Functions • Last Revised 30 Dec 1996

NAME	NAME getnetpath, setnetpath, endnetpath – get /etc/netconfig entry corresponding to NETPATH component		
SYNOPSIS	NOPSIS #include <netconfig.h></netconfig.h>		
	<pre>struct netconfig *getnetpath(void *handlep);</pre>		
	<pre>void *setnetpath(void);</pre>		
	<pre>int endnetpath(void *handlep);</pre>		
DESCRIPTION	The routines described on this page are part of the Network Selection component. They provide the application access to the system network configuration database, /etc/netconfig, as it is "filtered" by the NETPATH environment variable. See environ(5). See getnetconfig(3NSL) for other routines that also access the network configuration database directly. The NETPATH variable is a list of colon-separated network identifiers.		
	<pre>getnetpath() returns a pointer to the netconfig database entry corresponding to the first valid NETPATH component. The netconfig entry is formatted as a struct netconfig. On each subsequent call, getnetpath() returns a pointer to the netconfig entry that corresponds to the next valid NETPATH component. getnetpath() can thus be used to search the netconfig database for all networks included in the NETPATH variable. When NETPATH has been exhausted, getnetpath() returns NULL.</pre>		
	A call to setnetpath() "binds" to or "rewinds" NETPATH. setnetpath() must called before the first call to getnetpath() and may be called at any other time. returns a handle that is used by getnetpath().		
	getnetpath() silently ignores invalid NETPATH components. A NETPATH component is invalid if there is no corresponding entry in the netconfig data		
	If the NETPATH variable is unset, getnetpath() behaves as if NETPATH were set to the sequence of "default" or "visible" networks in the netconfig database, in the order in which they are listed.		
	<pre>endnetpath() may be called to "unbind" from NETPATH when processing is complete, releasing resources for reuse. Programmers should be aware, however, that endnetpath() frees all memory allocated by getnetpath() for the struct netconfig data structure. endnetpath() returns 0 on success and -1 on failure (for example, if setnetpath() was not called previously).</pre>		
RETURN VALUES	<pre>setnetpath() returns a handle that is used by getnetpath(). In case of an error, setnetpath() returns NULL.nc_perror() or nc_sperror() can be used to print out the reason for failure. See getnetconfig(3NSL).</pre>		
	When first called, getnetpath() returns a pointer to the netconfig database entry corresponding to the first valid NETPATH component. When NETPATH has been exhausted, getnetpath() returns NULL.		

### getnetpath(3NSL)

endnetpath() returns 0 on success and -1 on failure (for example, if setnetpath() was not called previously).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** getnetconfig(3NSL), netconfig(4), attributes(5), environ(5)

NAME	getpeername – get name of connected peer		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>int getpeername(int s,</pre>	struct soc	kaddr * <i>name</i> , socklen_t * <i>namelen</i> );
DESCRIPTION	getpeername() returns the name of the peer connected to socket <i>s</i> . The int pointed to by the <i>namelen</i> parameter should be initialized to indicate the amount of space pointed to by <i>name</i> . On return it contains the actual size of the name returned (in bytes), prior to any truncation. The name is truncated if the buffer provided is too small.		
RETURN VALUES	If successful, getpeername() returns 0; otherwise it returns -1 and sets errno to indicate the error.		
ERRORS	The call succeeds unless:		
	EBADF	The argume	nt <i>s</i> is not a valid descriptor.
	ENOMEM	There was ir complete.	nsufficient user memory for the operation to
	ENOSR		insufficient STREAMS resources available ation to complete.
	ENOTCONN	The socket is	s not connected.
	ENOTSOCK	The argume	nt <i>s</i> is not a socket.
ATTRIBUTES	<b>ATTRIBUTES</b> See attributes(5) for descriptions of the following attributes:		following attributes:
	ATTRIBUTE TYPE ATTRIBUTE		ATTRIBUTE VALUE
	MT-Level Safe		Safe
SEE ALSO accept(3SOCKET), bind(3SOCKET), ge socket(3SOCKET), attributes(5), so			

# getpeername(3XNET)

NAME	getpeername – get the name of the peer socket		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
<pre>int getpeername(int socket, struct sockaddr *address, socklen_ *address_len);</pre>			<pre>sockaddr *address, socklen_t</pre>
<b>DESCRIPTION</b> The getpeername() function retrieves the peer address of the specified so this address in the sockaddr structure pointed to by the <i>address</i> argument, the length of this address in the object pointed to by the <i>address_len</i> argument		nted to by the <i>address</i> argument, and stores	
	If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.		
	If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by <i>address</i> is unspecified.		
<b>RETURN VALUES</b>	Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The getpeername() function will fail if:		
	EBADF The <i>socket</i> argument is not a valid file descriptor.		
	EFAULT	The <i>address</i> or <i>address_le</i>	<i>en</i> parameter can not be accessed or written.
	EINVAL The socket has been shut down.		
	ENOTCONN The socket is not connected or otherwise has not had the peer prespecified.		
	ENOTSOCK The <i>socket</i> argument does not refer to a socket.		
	EOPNOTSUPP The operation is not supported for the socket protocol.		
	The getpeername() function may fail if:		
	ENOBUFS Insufficient resources were available in the system to complete the call.		
	ENOSR	There were insufficient operation to complete.	STREAMS resources available for the
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level MT-Safe		
	1		

<sup>178</sup> man pages section 3: Networking Library Functions • Last Revised 8 May 1998

getpeername(3XNET)

# getprotobyname(3SOCKET)

tprotodyname(SSOCKET)			
NAME	getprotobyname, getprotobyname_r, getprotobynumber, getprotobynumber_r, getprotoent, getprotoent_r, setprotoent, endprotoent – get protocol entry		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <netdb.h></netdb.h></pre>		
	<pre>struct protoent *getprotobyname(const char *name);</pre>		
	<pre>struct protoent *getprotobyname_r(const char *name, struct     protoent *result, char *buffer, int buflen);</pre>		
	<pre>struct protoent *getprotobynumber(int proto);</pre>		
	<pre>struct protoent *getprotobynumber_r(int proto, struct protoent     *result, char *buffer, int buflen);</pre>		
	<pre>struct protoent *getprotoent(void);</pre>		
	<pre>struct protoent *getprotoent_r(struct protoent *result, char *buffer,</pre>		
	<pre>int setprotoent(int stayopen);</pre>		
	<pre>int endprotoent(void);</pre>		
DESCRIPTION	These routines return a protocol entry. Two types of interfaces are supported: reentrant (getprotobyname_r(), getprotobynumber_r(), and getprotoent_r()) and non-reentrant (getprotobyname(), getprotobynumber(), and getprotoent()). The reentrant routines may be used in single-threaded applications and are safe for multi-threaded applications, making them the preferred interfaces.		
	The reentrant routines require additional parameters which are used to return results data. <i>result</i> is a pointer to a struct protoent structure and will be where the returned results will be stored. <i>buffer</i> is used as storage space for elements of the returned results. <i>buflen</i> is the size of <i>buffer</i> and should be large enough to contain all returned data. <i>buflen</i> must be at least 1024 bytes.		
	getprotobyname_r(),getprotobynumber_r(),and getprotoent_r() each return a protocol entry.		
	The entry may come from one of the following sources: the protocols file (see protocols(4)), the NIS maps "protocols.byname" and "protocols.bynumber", and the NIS+ table "protocols". The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4) for details). Some name services such as NIS will return only one name for a host, whereas others such as NIS+ or DNS will return all aliases.		
	getprotobyname_r() and getprotobynumber_r() sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until an EOF is encountered.		

180 man pages section 3: Networking Library Functions • Last Revised 25 Jul 2000

	getprotobymanic(oboccell)			
	getprotobyname() and getprotobynumber() have the same functionality as getprotobyname_r() and getprotobynumber_r() except that a static buffer is used to store returned results. These routines are unsafe in a multi-threaded application.			
	<pre>getprotoent_r() enumerates protocol entries: successive calls to getprotoent_r() will return either successive protocol entries or NULL. Enumeration may not be supported by some sources. Note that if multiple threads ca getprotoent_r(), each will retrieve a subset of the protocol database.</pre>			
	getprotent() has the same functionality as getprotent_r() except that a static buffer is used to store returned results. This routine is unsafe in a multi-threaded application.			
	<pre>setprotoent() "rewinds" to the beginning of the enumeration of protocol entries. If the stayopen flag is non-zero, resources such as open file descriptors are not deallocated after each call to getprotobynumber_r() and getprotobyname_r(). Calls to getprotobyname_r() , getprotobyname() , getprotobynumber_r() and getprotobynumber() may leave the enumeration in an indeterminate state, so setprotoent() should be called before the first getprotoent_r() or getprotoent(). Note that setprotoent() has process-wide scope, and "rewinds" the protocol entries for all threads calling getprotoent_r() as well as main-thread calls to getprotoent().</pre>			
	endprotoent() may be called to indicate that protocol processing is complete; the system may then close any open protocols file, deallocate storage, and so forth. It is legitimate, but possibly less efficient, to call more protocol routines after endprotoent().			
	The internal representation of a protocol entry is a protoent structure defined in <netdb.h> with the following members:</netdb.h>			
	<pre>char *p_name; char **p_aliases; int p_proto;</pre>			
RETURN VALUES	getprotobyname_r(), getprotobyname(), getprotobynumber_r(), and getprotobynumber() return a pointer to a struct protoent if they successfully locate the requested entry; otherwise they return NULL.			
	getprotoent_r() and getprotoent() return a pointer to a struct protoent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.			
ERRORS	getprotobyname_r( ), getprotobynumber_r( ), and getprotoent_r() will fail if the following is true:			
	ERANGE length of the buffer supplied by caller is not large enough to store the result.			

#### getprotobyname(3SOCKET)

### FILES | /etc/protocols

/etc/nsswitch.conf

**ATTRIBUTES** 

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

### **SEE ALSO** intro(3), nsswitch.conf(4), protocols(4), attributes(5), netdb(3HEAD)

**NOTES** Although getprotobyname\_r(), getprotobynumber\_r(), and getprotoent\_r() are not mentioned by POSIX 1003.1c, they were added to complete the functionality provided by similar thread-safe functions.

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

The routines getprotobyname\_r(), getprotobynumber\_r(), and getprotoent\_r() are reentrant and multi-thread safe. The reentrant interfaces can be used in single-threaded as well as multi-threaded applications and are therefore the preferred interfaces.

The routines getprotobyname(), getprotobyaddr(), and getprotoent() use static storage, so returned data must be copied if it is to be saved. Because of their use of static storage for returned data, these routines are not safe for multi-threaded applications.

setprotoent() and endprotoent() have process-wide scope, and are therefore
not safe in multi-threaded applications.

Use of getprotoent\_r() and getprotoent() is discouraged; enumeration is well-defined for the protocols file and is supported (albeit inefficiently) for NIS and NIS+, but in general may not be well-defined. The semantics of enumeration are discussed in nsswitch.conf(4).

**BUGS** Only the Internet protocols are currently understood.

Programs that call getprotobyname\_r() or getprotobynumber\_r() routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.

NAME	getpublickey, getsecretkey, publickey – retrieve public or secret key			
SYNOPSIS	<pre>#include <rpc rpc.h=""> #include <rpc key_prot.h=""></rpc></rpc></pre>			
	<pre>int getpublickey(const char netname[MAXNETNAMELEN], char     publickey[HEXKEYBYTES+1]);</pre>			
	<pre>int getsecretkey(const char netname[MAXNETNAMELEN], char secretkey[HEXKEYBYTES+1], const char *passwd);</pre>			
DESCRIPTION	getpublickey() and getsecretkey() get public and secret keys for <i>netname</i> . The key may come from one of the following sources: the /etc/publickey file (see publickey(4)) or the NIS map "publickey.byname" or the NIS+ table "cred.org_dir". The sources and their lookup order are specified in the /etc/nsswitch.conf file (see nsswitch.conf(4)).			
	getsecretkey() has an extra argument, passwd, used to decrypt the encrypted secret key stored in the database.			
RETURN VALUES	Both routines return 1 if they are successful in finding the key, 0 otherwise. The keys are returned as NULL-terminated, hexadecimal strings. If the password supplied to getsecretkey() fails to decrypt the secret key, the routine will return 1 but the <i>secretkey</i> [0] will be set to NULL.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SEE ALSO** secure\_rpc(3NSL), nsswitch.conf(4), publickey(4), attributes(5)

**WARNINGS** If getpublickey() gets the public key from any source other than NIS+, all authenticated NIS+ operations may fail. To ensure that this does not happen, edit the nsswitch.conf(4) file to make sure that the public key is obtained from NIS+.

getrpcbyname(3NSL)

NAME	getrpcbyname, getrpcbyname_r, getrpcbynumber, getrpcbynumber_r, getrpcent, getrpcent_r, setrpcent, endrpcent – get RPC entry		
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpc rpcent.h=""></rpc></pre>		
	<pre>struct rpcent *getrpcbyname(const char *name);</pre>		
	<pre>struct rpcent *getrpcbyname_r(const char *name, struct rpcent     *result, char *buffer, int buflen);</pre>		
	<pre>struct rpcent *getrpcbynumber(const int number);</pre>		
	<pre>struct rpcent *getrpcbynumber_r(const int number, struct rpcent</pre>		
	<pre>struct rpcent *getrpcent(void);</pre>		
	<pre>struct rpcent *getrpcent_r(struct rpcent *result, char *buffer, int</pre>		
	<pre>void setrpcent(const int stayopen);</pre>		
	<pre>void endrpcent(void);</pre>		
DESCRIPTION	These functions are used to obtain entries for RPC (Remote Procedure Call) services. An entry may come from any of the sources for rpc specified in the /etc/nsswitch.conf file (see nsswitch.conf(4)).		
	getrpcbyname() searches for an entry with the RPC service name specified by the parameter <i>name</i> .		
	getrpcbynumber() searches for an entry with the RPC program number <i>number</i> .		
	The functions setrpcent(), getrpcent(), and endrpcent() are used to enumerate RPC entries from the database.		
	<pre>setrpcent() sets (or resets) the enumeration to the beginning of the set of RPC entries. This function should be called before the first call to getrpcent(). Calls to getrpcbyname() and getrpcbynumber() leave the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endrpcent().</pre>		
	Successive calls to getrpcent() return either successive entries or NULL, indicating the end of the enumeration.		
	endrpcent() may be called to indicate that the caller expects to do no further RPC entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more RPC entry retrieval functions after calling endrpcent().		

184 man pages section 3: Networking Library Functions • Last Revised 20 Feb 1998

## Reentrant The functions getrpcbyname(), getrpcbynumber(), and getrpcent() use static Interfaces storage that is re-used in each call, making these routines unsafe for use in multithreaded applications. The functions getrpcbyname r(), getrpcbynumber r(), and getrpcent r()provide reentrant interfaces for these operations. Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "\_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications. Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *result* must be a pointer to a struct rpcent structure allocated by the caller. On successful completion, the function returns the RPC entry in this structure. The parameter *buffer* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the RPC entry data. All of the pointers within the returned struct rpcent result point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the RPC entry. The parameter buflen should give the size in bytes of the buffer indicated by *buffer*. For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. setrpcent() may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to get r(), the threads will enumerate disjoint subsets of the RPC entry database. Like their non-reentrant counterparts, getrpcbyname r() and getrpcbynumber r() leave the enumeration position in an indeterminate state. **RETURN VALUES** RPC entries are represented by the struct rpcent structure defined in <rpc/rpcent.h>: struct rpcent { char \*r\_name; /\* name of this rpc service char \*\*r\_aliases; /\* zero-terminated list of alternate names \*/ int r\_number; /\* rpc program number \*/ }; The functions getrpcbyname(), getrpcbyname\_r(), getrpcbynumber(), and getrpcbynumber r() each return a pointer to a struct rpcent if they successfully locate the requested entry; otherwise they return NULL. The functions getrpcent() and getrpcent r() each return a pointer to a struct rpcent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

getrpcbyname(3NSL	.)		
	The functions getrpcbyname(), getrpcbynumber(), and getrpcent() use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.		
	When the pointer returned by the reentrant functions getrpcbyname_r(), getrpcbynumber_r(), and getrpcent_r() is non-NULL, it is always equal to the <i>result</i> pointer that was supplied by the caller.		
ERRORS	The reentrant functions getrpcyname_r(), getrpcbynumber_r() and getrpcent_r() will return NULL and set errno to ERANGE if the length of the buffer supplied by caller is not large enough to store the result. See intro(2) for the proper usage and interpretation of errno in multithreaded applications.		
FILES	/etc/rpc		
	/etc/nsswitch.conf		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	See "Reentrant Interfaces" in DESCRIPTION.	
SEE ALSO	rpcinfo(1M), rpc(3NSL), nsswitch.cor	nf(4), rpc(4), attributes(5)	
WARNINGS	The reentrant interfaces getrpcbyname_r(), getrpcbynumber_r(), and getrpcent_r() are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.		
NOTES	Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.		
	When compiling multithreaded applications, see intro(3), Notes On Multithreaded Applications, for information about the use of the _REENTRANT flag.		
	Use of the enumeration interfaces getrpcent() and getrpcent_r() is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in nsswitch.conf(4).		

186 man pages section 3: Networking Library Functions • Last Revised 20 Feb 1998

NAME	getservbyname, getservbyname_r, getservbyport, getservbyport_r, getservent, getservent_r, setservent, endservent – get service entry				
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <netdb.h></netdb.h></pre>				
	<pre>struct servent *getservbyname(const char *name, const char *proto);</pre>				
	<pre>struct servent *getservbyname_r(const char *name, const char *proto, struct servent *result, char *buffer, int buflen);</pre>				
	<pre>struct servent *getservbyport(int port, const char *proto);</pre>				
	<pre>struct servent *getservbyport_r(int port, const char *proto, struct servent *result, char *buffer, int buflen);</pre>				
	<pre>struct servent *getservent(void);</pre>				
	<pre>struct servent *getservent_r(struct servent *result, char *buffer, int buflen);</pre>				
	<pre>int setservent(int stayopen);</pre>				
	<pre>int endservent(void);</pre>				
DESCRIPTION	These functions are used to obtain entries for Internet services. An entry may come from any of the sources for services specified in the /etc/nsswitch.conf file. See nsswitch.conf(4).				
	getservbyname() and getservbyport() sequentially search from the beginning of the file until a matching protocol name or port number is found, or until end-of-file is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.				
	getservbyname() searches for an entry with the Internet service name specified by the parameter <i>name</i> .				
	getservbyport() searches for an entry with the Internet port number port.				
	All addresses are returned in network order. In order to interpret the addresses, byteorder(3SOCKET)				
	must be used for byte order conversion. The string <i>proto</i> is used by both getservbyname() and getservbyport() to restrict the search to entries with the specified protocol. If <i>proto</i> is NULL, entries with any protocol may be returned.				
	The functions setservent(), getservent(), and endservent() are used to enumerate entries from the services database.				
	<pre>setservent() sets (or resets) the enumeration to the beginning of the set of service entries. This function should be called before the first call to getservent(). Calls to the functions getservbyname() and getservbyport() leave the enumeration</pre>				

# getservbyname(3SOCKET)

ervbyname(350	CRE1)
	position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endservent().
	<pre>getservent() reads the next line of the file, opening the file if necessary. getservent() opens and rewinds the file. If the <i>stayopen</i> flag is non-zero, the net data base will not be closed after each call to getservent() (either directly, or indirectly through one of the other "getserv" calls).</pre>
	Successive calls to $getservent()$ return either successive entries or NULL, indicating the end of the enumeration.
	endservent() closes the file. endservent() may be called to indicate that the caller expects to do no further service entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more service entry retrieval functions after calling endservent().
Reentrant Interfaces	The functions getservbyname(), getservbyport(), and getservent() use static storage that is re-used in each call, making these functions unsafe for use in multithreaded applications.
	The functions getservbyname_r(), getservbyport_r(), and getservent_r() provide reentrant interfaces for these operations.
	Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "_r" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.
	Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter <i>result</i> must be a pointer to a struct servent structure allocated by the caller. On successful completion, the function returns the service entry in this structure. The parameter <i>buffer</i> must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the service entry data. All of the pointers within the returned struct servent <i>result</i> point to data stored within this buffer. See the RETURN VALUES section of this man page. The buffer must be large enough to hold all of the data associated with the service entry. The parameter <i>buflen</i> should give the size in bytes of the buffer indicated by <i>buffer</i> .
	For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. setservent() may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to getservent_r(), the threads will enumerate disjoint subsets of the service database.
	Like their non-reentrant counterparts, getservbyname_r() and getservbyport_r() leave the enumeration position in an indeterminate state.

#### getservbyname(3SOCKET)

```
RETURN VALUES
                     Service entries are represented by the struct servent structure defined in
                     <netdb.h>:
                     struct servent {
                         uct servenc {
    char *s_name; /* official nam
    char **s_aliases; /* alias list */
        . /* port service
                                                       /* official name of service */
                         int s_port;
                                                    /* port service resides at */
/* protocol to use */
                         char
                                  *s_proto;
                     };
                     The members of this structure are:
                     s name
                                        The official name of the service.
                                        A zero terminated list of alternate names for the service.
                     s aliases
                                        The port number at which the service resides. Port numbers are
                     s port
                                        returned in network byte order.
                                        The name of the protocol to use when contacting the service
                     s proto
                     The functions getservbyname(), getservbyname r(), getservbyport(),
                     and getservbyport r() each return a pointer to a struct servent if they
                     successfully locate the requested entry; otherwise they return NULL.
                     The functions getservent() and getservent r() each return a pointer to a
                     struct servent if they successfully enumerate an entry; otherwise they return
                     NULL, indicating the end of the enumeration.
                     The functions getservbyname(), getservbyport(), and getservent() use
                     static storage, so returned data must be copied before a subsequent call to any of these
                     functions if the data is to be saved.
                     When the pointer returned by the reentrant functions getservbyname r(),
                     getservbyport r(), and getservent r() is non-null, it is always equal to the
                     result pointer that was supplied by the caller.
         ERRORS
                     The reentrant functions getservbyname r(), getservbyport r() and
                     getservent r() will return NULL and set errno to ERANGE if the length of the
                     buffer supplied by caller is not large enough to store the result. See intro(2) for the
                     proper usage and interpretation of errno in multithreaded applications.
            FILES
                     /etc/services
                                                   Internet network services
                     /etc/netconfig
                                                   network configuration file
                     /etc/nsswitch.conf
                                                   configuration file for the name-service switch
    ATTRIBUTES
                     See attributes(5) for descriptions of the following attributes:
```

# getservbyname(3SOCKET)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	See "Reentrant Interfaces" in DESCRIPTION.	
SEE ALSO	<pre>intro(2), intro(3), byteorder(3SOCKET), netdir(3NSL), netconfig(4), nsswitch.conf(4), services(4), attributes(5), netdb(3HEAD)</pre>		
WARNINGS	The reentrant interfaces getservbyname_r(), getservbyport_r(), and getservent_r() are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.		
NOTES	The functions that return struct servent return the least significant 16-bits of the <i>s_port</i> field in <i>network byte order</i> . getservbyport() and getservbyport_r() also expect the input parameter <i>port</i> in the <i>network byte order</i> . See htons(3SOCKET) for more details on converting between host and network byte orders.		
	Programs that use the interfaces described is statically since the implementations of these linking of shared objects at run time.		
	In order to ensure that they all return consistent results, getservbyname(), getservbyname_r(), and netdir_getbyname() are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the inet family entries in netconfig(4) and the services: entry in nsswitch.conf(4). Similarly, getservbyport(), getservbyport_r(), and netdir_getbyaddr() are implemented in terms of the same internal library function. If the inet family entries in netconfig(4) have a "-" in the last column for nametoaddr libraries, then the entry for services in nsswitch.conf will be used; otherwise the nametoaddr libraries in that column will be used, and nsswitch.conf will not be consulted.		
	There is no analogue of getservent() and functions, so these enumeration functions geneswitch.conf. Thus enumeration may need by getservbyname(), getserv getservbyport_r().	o straight to the services entry in return results from a different source than	
	When compiling multithreaded application <i>Applications</i> , for information about the use of		
	Use of the enumeration interfaces getserv discouraged; enumeration may not be supp semantics of enumeration are discussed fur	ported for all database sources. The	

NAME	getsockname – ge	et socket name	
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>int getsockname(int s, struct sockaddr *name, socklen_t *namelen);</pre>		
DESCRIPTION	getsockname() returns the current <i>name</i> for socket <i>s</i> . The <i>namelen</i> parameter should be initialized to indicate the amount of space pointed to by <i>name</i> . On return it contains the actual size in bytes of the name returned.		
RETURN VALUES	If successful, get indicate the error	sockname() returns 0; otherwise it returns -1 and sets <i>errno</i> to	
ERRORS	The call succeeds	s unless:	
	EBADF	The argument <i>s</i> is not a valid file descriptor.	
	ENOMEM	There was insufficient memory available for the operation to complete.	
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.	
	ENOTSOCK	The argument <i>s</i> is not a socket.	
	See attributes(5) for descriptions of the following attributes:		
ATTRIBUTES	See attributes	s(5) for descriptions of the following attributes:	
ATTRIBUTES		s(5) for descriptions of the following attributes:           TRIBUTE TYPE         ATTRIBUTE VALUE	
ATTRIBUTES			
ATTRIBUTES SEE ALSO	AT MT-Level	TRIBUTE TYPE ATTRIBUTE VALUE	

# getsockname(3XNET)

NAME	getsockname – get the socket name			
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>			
	<pre>int getsockname(int socket, struct sockaddr *address, socklen_t     *address_len);</pre>			
DESCRIPTION	The getsockname() function retrieves the locally-bound name of the specified socket, stores this address in the sockaddr structure pointed to by the <i>address</i> argument, and stores the length of this address in the object pointed to by the <i>address_len</i> argument.			
	If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.			
	If the socket has n to by <i>address</i> is un		name, the value stored in the object pointed	
RETURN VALUES	Upon successful completion, 0 is returned, the <i>address</i> argument points to the address of the socket, and the <i>address_len</i> argument points to the length of the address. Otherwise, -1 is returned and errno is set to indicate the error.			
ERRORS	The getsocknam	e() function will fail:		
	EBADF	The <i>socket</i> argument is n	not a valid file descriptor.	
	EFAULT	The <i>address</i> or <i>address_len</i> parameter can not be accessed or written.		
	ENOTSOCK	The <i>socket</i> argument does not refer to a socket.		
	EOPNOTSUPP	The operation is not sup	pported for this socket's protocol.	
	The getsockname () function may fail if:			
	EINVAL	The socket has been shu	ıt down.	
	ENOBUFS	Insufficient resources we call.	ere available in the system to complete the	
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.		
ATTRIBUTES	YES       See attributes(5) for descriptions of the following attributes:         ATTRIBUTE TYPE       ATTRIBUTE VALUE         MT-Level       MT-Safe		following attributes:	
			ATTRIBUTE VALUE	
			MT-Safe	
SEE ALSO	accept(3XNET), attributes(5)	bind(3XNET),getpeerr	name(3XNET), socket(3XNET)	

NAME	getsockopt, setsockopt – get and set options on sockets					
SYNOPSIS	<pre>cc [ flag ] filelsoc #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>	cket -lnsl [ <i>library</i> ]				
	<pre>int getsockopt(int s, int level, int optname, void *optval, int *optlen);</pre>					
	<pre>int setsockopt(int s, i</pre>	<pre>int setsockopt(int s, int level, int optname, const void *optval, int</pre>				
DESCRIPTION	getsockopt() and setsockopt() manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.					
	When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, <i>level</i> is specified as SOL_SOCKET. To manipulate options at any other level, <i>level</i> is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, <i>level</i> is set to the TCP protocol number . See getprotobyname(3SOCKET).					
	The parameters <i>optval</i> and <i>optlen</i> are used to access option values for setsockop 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 ret to indicate the actual size of the value returned. Use a 0 <i>optval</i> if no option value is be supplied or returned.					
	<i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file < <sys socket.h=""> contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</sys>					
	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. struct linger is defined in < <sys socket.h="">. struct linger contains the following members:</sys>					
	l_onoff	on = $1/\text{off} = 0$				
	l_linger linger time, in seconds					
	The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt().					
	SO_DEBUG	enable/disable recording of debugging information				
	SO_REUSEADDR	enable/disable local address reuse				
	SO_KEEPALIVE	enable/disable keep connections alive				
	1	<b>.</b>				

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\_KEEPALIVE 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.

SO\_SNDBUF and SO\_RCVBUF are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. The maximum buffer size for UDP is determined by the value of the ndd variable udp\_max\_buf. The maximum buffer size for TCP is determined the value of the ndd variable tcp\_max\_buf. Use the ndd(1M) utility to determine the current default values. See the *Solaris Tunable Parameters Reference Manual* for information on setting the values of udp\_max\_buf and tcp\_max\_buf.

	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 ( <i>to</i> 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. If successful, getsockopt() returns 0; otherwise, it returns -1 and sets errno to		
RETURN VALUES	If successful, getsockopt () indicate the error.	returns 0; oth	terwise, it returns $-1$ and sets errno to
ERRORS	The call succeeds unless:		
	EBADF	The argume	nt <i>s</i> is not a valid file descriptor.
	ENOMEM	There was ir operation to	nsufficient memory available for the complete.
	ENOPROTOOPT	The option is	s unknown at the level indicated.
	ENOSR		nsufficient STREAMS resources available ation to complete.
	ENOTSOCK	The argume	nt <i>s</i> is not a socket.
	ENOBUFS	SO_SNDBUF	or SO_RCVBUF exceeds a system limit.
	EINVAL	Invalid leng	th for IP_OPTIONS.
	EHOSTUNREACH	Invalid addr	ress for IP_MULTICAST_IF.
	EINVAL		ast address for IP_ADD_MEMBERSHIP and EMBERSHIP.
	EADDRNOTAVAIL	Bad interface address for IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP.	
	EADDRINUSE	Address alre	eady joined for IP_ADD_MEMBERSHIP.
	ENOENT	Address not	joined for IP_DROP_MEMBERSHIP.
	EPERM	No permissi	ons.
ATTRIBUTES	See attributes(5) for descri	iptions of the	following attributes:
	ATTRIBUTE TYPE		ATTRIBUTE VALUE

Safe

MT-Level

## getsockopt(3SOCKET)

SEE ALSO | ndd(1M), close(2), ioctl(2), read(2), bind(3SOCKET), getprotobyname(3SOCKET), recv(3SOCKET), send(3SOCKET), socket(3SOCKET), attributes(5)

Solaris Tunable Parameters Reference Manual

196 man pages section 3: Networking Library Functions • Last Revised 24 Aug 2001

		8
NAME	getsockopt – get the socket op	ptions
SYNOPSIS	<pre>cc [ flag ] filelxnd #include <sys socket.h=""></sys></pre>	et [ <i>library</i> ]
	<pre>int getsockopt(int socka socklen_t *option_ler</pre>	<pre>et, int level, int option_name, void *option_value, );</pre>
DESCRIPTION	<pre>option_name argument for the option value is greater than op option_value argument will be</pre>	retrieves the value for the option specified by the socket specified by the <i>socket</i> argument. If the size of the <i>vtion_len</i> , the value stored in the object pointed to by the silently truncated. Otherwise, the object pointed to by one modified to indicate the actual length of the value.
	options at the socket level, sp options at other levels, supply controlling the option. For ex- the TCP (Transport Control P.	he protocol level at which the option resides. To retrieve ecify the <i>level</i> argument as SOL_SOCKET. To retrieve y the appropriate protocol number for the protocol ample, to indicate that an option will be interpreted by rotocol), set <i>level</i> to the protocol number of TCP, as . h> header, or as determined by using unction.
	The socket in use may require getsockopt() function.	e the process to have appropriate privileges to use the
	The <i>option_name</i> argument specifies a single option to be retrieved. It can be one of the following values defined in <sys socket.h="">:</sys>	
	SO_DEBUG	Reports whether debugging information is being recorded. This option stores an int value. This is a boolean option.
	SO_ACCEPTCONN	Reports whether socket listening is enabled. This option stores an int value.
	SO_BROADCAST	Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option stores an int value. This is a boolean option.
	SO_REUSEADDR	Reports whether the rules used in validating addresses supplied to bind(3XNET) should allow reuse of local addresses, if this is supported by the protocol. This option stores an int value. This is a boolean option.
	SO_KEEPALIVE	Reports whether connections are kept active with periodic transmission of messages, if this is supported by the protocol.
		If the connected socket fails to respond to these messages, the connection is broken and processes

# getsockopt(3XNET)

geisockopi(SANET)		
		writing to that socket are notified with a SIGPIPE signal. This option stores an int value.
		This is a boolean option.
	SO_LINGER	Reports whether the socket lingers on close(2) if data is present. If SO_LINGER is set, the system blocks the process during close(2) until it can transmit the data or until the end of the interval indicated by the l_linger member, whichever comes first. If SO_LINGER is not specified, and close(2) is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option stores a linger structure.
	SO_OOBINLINE	Reports whether the socket leaves received out-of-band data (data marked urgent) in line. This option stores an int value. This is a boolean option.
	SO_SNDBUF	Reports send buffer size information. This option stores an int value.
	SO_RCVBUF	Reports receive buffer size information. This option stores an int value.
	SO_ERROR	Reports information about error status and clears it. This option stores an int value.
	SO_TYPE	Reports the socket type. This option stores an int value.
	SO_DONTROUTE	Reports whether outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option stores an int value. This is a boolean option.
	For boolean options, a zero va value indicates that the option	lue indicates that the option is disabled and a non-zero n is enabled.
	Options at other protocol leve	ls vary in format and name.
	The socket in use may require getsockopt() function.	the process to have appropriate privileges to use the
RETURN VALUES	Upon successful completion, g errno is set to indicate the er	getsockopt() returns 0. Otherwise, -1 is returned and ror.
ERRORS	The getsockopt() function	will fail if:

198 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

ATTRIBUTE TYPE     ATTRIBUTE VALUE       MT-Level     MT-Safe				getsockopt(3ANE)
EINVAL       The specified option is invalid at the specified socket level.         ENOPROTOOPT       The option is not supported by the protocol.         ENOTSOCK       The socket argument does not refer to a socket.         The getsockopt() function may fail if:       EACCES         EACCES       The calling process does not have the appropriate privileges.         EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		EBADF	The socket argument is	not a valid file descriptor.
ENOPROTOOPT       The option is not supported by the protocol.         ENOTSOCK       The socket argument does not refer to a socket.         The getsockopt()       function may fail if:         EACCES       The calling process does not have the appropriate privileges.         EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         Image: Attribute type       Attribute value         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		EFAULT		ion_len parameter can not be accessed or
ENOTSOCK       The socket argument does not refer to a socket.         The getsockopt () function may fail if:       EACCES         EACCES       The calling process does not have the appropriate privileges.         EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		EINVAL	The specified option is	invalid at the specified socket level.
The getsockopt () function may fail if:         EACCES       The calling process does not have the appropriate privileges.         EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		ENOPROTOOPT	The option is not supp	orted by the protocol.
EACCES       The calling process does not have the appropriate privileges.         EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		ENOTSOCK	The <i>socket</i> argument do	bes not refer to a socket.
EINVAL       The socket has been shut down.         ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         Image: Attribute system to complete type       Attribute value         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		The getsockopt	c() function may fail if:	
ENOBUFS       Insufficient resources are available in the system to complete the call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         Image: Attribute type       Attribute value         MT-Level       MT-Safe         SEE ALSO       close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		EACCES	The calling process doe	es not have the appropriate privileges.
call.         ENOSR       There were insufficient STREAMS resources available for the operation to complete.         See attributes(5) for descriptions of the following attributes:         ATTRIBUTE TYPE         ATTRIBUTE TYPE         MT-Level         MT-Level         Close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		EINVAL	The socket has been sh	ut down.
operation to complete.         TRIBUTES       See attributes(5) for descriptions of the following attributes:         ATTRIBUTE TYPE       ATTRIBUTE VALUE         MT-Level       MT-Safe		ENOBUFS		re available in the system to complete the
ATTRIBUTE TYPE     ATTRIBUTE VALUE       MT-Level     MT-Safe       SEE ALSO     close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),		ENOSR		STREAMS resources available for the
MT-Level     MT-Safe       SEE ALSO     close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),	TRIBUTES	See attributes	s(5) for descriptions of the	e following attributes:
SEE ALSO close(2), bind(3XNET), endprotoent(3XNET), setsockopt(3XNET),				
		ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	SEE ALSO	MT-Level		MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe
	SEE ALSO	MT-Level	3XNET), endprotoent(3	MT-Safe

NAME	gss_accept_sec_context - accept a security context initiated by a peer application
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>
	<pre>OM_uint32 gss_accept_sec_context(OM_uint32 *minor_status, gss_ctx_id_t *context_handle, const gss_cred_id_t acceptor_cred_handle, const gss_buffer_t input_token, const gss_channel_bindings_t input_chan_bindings, const gss_name_t *src_name, gss_OID *mech_type, gss_buffer_t output_token, OM_uint32 *ret_flags, OM_uint32 *time_rec, gss_cred_id_t *delegated_cred_handle);</pre>
DESCRIPTION	The gss_accept_sec_context() function allows a remotely initiated security context between the application and a remote peer to be established. The routine may return an <i>output_token</i> , which should be transferred to the peer application, where the peer application will present it to gss_init_sec_context(). See gss_init_sec_context(3GSS). If no token need be sent, gss_accept_sec_context() will indicate this by setting the length field of the <i>output_token</i> argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, gss_accept_sec_context() will return a status flag of GSS_S_CONTINUE_NEEDED, in which case it should be called again when the reply token is received from the peer application, passing the token to gss_accept_sec_context() by means of the <i>input_token</i> parameters.
	Portable applications should be constructed to use the token length and return status to determine whether to send or to wait for a token.
	Whenever gss_accept_sec_context() returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output parameters:
	<ul> <li>The value returned by means of the <i>time_rec</i> parameter is undefined.</li> </ul>
	<ul> <li>Unless the accompanying <i>ret_flags</i> parameter contains the bit GSS_C_PROT_READY_FLAG, which indicates that per-message services may be applied in advance of a successful completion status, the value returned by the <i>mech_type</i> parameter may be undefined until gss_accept_sec_context() returns a major status value of GSS_S_COMPLETE.</li> </ul>
	The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned by means of the <i>ret_flags</i> parameter are values that would be valid if context establishment were to succeed.
	The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within <i>ret_flags</i> indicate the actual state at the time gss_accept_sec_context() returns, whether or not the context is fully established. However, applications should not rely on this behavior, as GSS_C_PROT_READY_FLAG was not defined in Version 1 of the GSS-API. Instead, applications should be prepared to use per-message services after a

successful context establishment, based upon the GSS\_C\_INTEG\_FLAG and GSS\_C\_CONF\_FLAG values.

All other bits within the *ret\_flags* argument are set to zero.

While gss\_accept\_sec\_context() returns GSS\_S\_CONTINUE\_NEEDED, the values returned by means of the the *ret\_flags* argument indicate the services available from the established context. If the initial call of gss\_accept\_sec\_context() fails, no context object is created, and the value of the *context\_handle* parameter is set to GSS\_C\_NO\_CONTEXT. In the event of a failure on a subsequent call, the security context and the *context\_handle* parameter are left untouched for the application to delete using gss\_delete\_sec\_context(3GSS). During context establishment, the informational status bits GSS\_S\_OLD\_TOKEN and GSS\_S\_DUPLICATE\_TOKEN indicate fatal errors; GSS-API mechanisms always return them in association with a routine error of GSS\_S\_FAILURE. This pairing requirement did not exist in version 1 of the GSS-API specification, so applications that wish to run over version 1 implementations must special-case these codes.

#### PARAMETERS

The parameter descriptions for gss accept sec context() follow:

minor_status	The status code returned by the underlying mechanism.
context_handle	The context handle to return to the initiator. This should be set to GSS_C_NO_CONTEXT before the loop begins.
acceptor_cred_handle	The handle for the credentials acquired by the acceptor, typically through gss_acquire_cred(). It may be initialized to GSS_C_NO_CREDENTIAL to indicate a default credential to use. If no default credential is defined, the function returns GSS_C_NO_CRED.
input_token_buffer	Token received from the context initiative.
input_chan_bindings	Optional application-specified bindings. Allows application to securely bind channel identification information to the security context. Set to GSS_C_NO_CHANNEL_BINDINGS if you do not want to use channel bindings.
src_name	The authenticated name of the context initiator. After use, this name should be deallocated by passing it to gss_release_name(). See gss_release_name(3GSS). If not required, specify NULL.
mech_type	The security mechanism used. Set to NULL if it does not matter which mechanism is used.
output_token	The token to send to the acceptor. Initialize it to GSS_C_NO_BUFFER before the function is called (or its length field set to zero). If the length is zero, no token need be sent.
ret_flags	Contains various independent flags, each of which indicates that the context supports a specific service option. If not

needed, specify NULL. Test the returned bit-mask *ret\_flags* value against its symbolic name to determine if the given option is supported by the context. *ret\_flags* may contain one of the following values:

#### GSS C DELEG FLAG

If true, delegated credentials are available by means of the *delegated\_cred\_handle* parameter. If false, no credentials were delegated.

GSS\_C\_MUTUAL\_FLAG

If true, a remote peer asked for mutual authentication. If false, no remote peer asked for mutual authentication.

#### GSS\_C\_REPLY\_FLAG

If true, replay of protected messages will be detected. If false, replayed messages will not be detected.

GSS C SEQUENCE FLAG

If true, out of sequence protected messages will be detected. If false, they will not be detected.

#### GSS C CONF FLAG

If true, confidentiality service may be invoked by calling the gss\_wrap() routine. If false, no confidentiality service is available by means of gss\_wrap().gss\_wrap() will provide message encapsulation, data-origin authentication and integrity services only.

GSS\_C\_INTEG\_FLAG

If true, integrity service may be invoked by calling either the gss\_get\_mic(3GSS) or the gss\_wrap(3GSS) routine. If false, per-message integrity service is not available.

GSS\_C\_ANON\_FLAG

If true, the initiator does not wish to be authenticated. The *src\_name* parameter, if requested, contains an anonymous internal name. If false, the initiator has been authenticated normally.

#### GSS C PROT READY FLAG

If true, the protection services specified by the states of GSS\_C\_CONF\_FLAG and GSS\_C\_INTEG\_FLAG are available if the accompanying major status return value is either GSS\_S\_COMPLETE or GSS\_S\_CONTINUE\_NEEDED. If false, the protection services are available only if the accompanying major status return value is GSS\_S\_COMPLETE.

			goo_accept_certac.a(c coo)
		other process gss_export	FLAG sultant security context may be transferred to es by means of a call to _sec_context(3GSS). If false, the security of be transferred.
	time_rec		ections for which the context will remain ILL if not required.
	delegated_cred_handle	context's initiato that the acceptor	alue for credentials received from the or. It is valid only if the initiator has requested act as a proxy: that is, if the <i>ret_flag</i> es to GSS_C_DELEG_FLAG.
<b>RETURN VALUES</b>	gss_accept_sec_co	ntext() may ret	urn the following status codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_CONTINUE_NE	EDED	A token from the peer application is required to complete the context, and that gss_accept_sec_context() must be called again with that token.
	GSS_S_DEFECTIVE_T	OKEN	Consistency checks performed on the <i>input_token</i> failed.
	GSS_S_DEFECTIVE_C	REDENTIAL	Consistency checks performed on the credential failed.
	GSS_S_NO_CRED		The supplied credentials were not valid for context acceptance, or the credential handle did not reference any credentials.
	GSS_S_CREDENTIALS	_EXPIRED	The referenced credentials have expired.
	GSS_S_BAD_BINDING	S	The <i>input_token</i> contains different channel bindings than those specified by means of the <i>input_chan_bindings</i> parameter.
	GSS_S_NO_CONTEXT		The supplied context handle did not refer to a valid context.
	GSS_S_BAD_SIG		The <i>input_token</i> contains an invalid MIC.
	GSS_S_OLD_TOKEN		The <i>input_token</i> was too old. This is a fatal error while establishing context.
	GSS_S_DUPLICATE_T	OKEN	The <i>input_token</i> is valid, but it is duplicate of a token already processed. This is a fatal error while establishing context.
	GSS_S_BAD_MECH		The token received specified a mechanism that is not supported by the implementation or the provided credential.

GSS S FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the minor\_status parameter details the error condition. **EXAMPLES EXAMPLE 1** Invoking gss\_accept\_sec\_context() Within a Loop A typical portable caller should always invoke gss accept sec context() within a loop: gss\_ctx\_id\_t context\_hdl = GSS\_C\_NO\_CONTEXT; do { receive\_token\_from\_peer(input\_token); maj stat = gss accept sec context(&min stat, &context\_hdl, cred hdl, input token, input\_bindings, &client\_name, &mech\_type, output\_token, &ret\_flags, &time\_rec, &deleg cred); if (GSS ERROR(maj stat)) { report\_error(maj\_stat, min\_stat); }; if (output token->length != 0) { send token to peer(output token); gss\_release\_buffer(&min\_stat, output\_token); }; if (GSS\_ERROR(maj\_stat)) { if (context hdl != GSS C NO CONTEXT) gss delete sec context(&min stat, &context\_hdl, GSS\_C\_NO\_BUFFER); break; }; } while (maj stat & GSS S CONTINUE NEEDED);

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

<sup>204</sup> man pages section 3: Networking Library Functions • Last Revised 18 Apr 2000

SEE ALSO gss\_delete\_sec\_context(3GSS), gss\_export\_sec\_context(3GSS), gss\_get\_mic(3GSS), gss\_init\_sec\_context(3GSS), gss\_release\_name(3GSS), gss\_wrap(3GSS), attributes(5)

GSS-API Programming Guide

## gss\_acquire\_cred(3GSS)

NAME	gss_acquire_cred – ac	equire a handle for a pre-existing credential by name
SYNOPSIS	cc -flag <i>file</i> #include <gssapi gs<="" th=""><th>lgss [ library ] ssapi.h&gt;</th></gssapi>	lgss [ library ] ssapi.h>
	<pre>OM_uint32 gss_acquire_cred (OM_uint32 *minor_status, const gss_name_t</pre>	
DESCRIPTION	pre-existing credentia the network; a function	cred() function allows an application to acquire a handle for a al by name. This routine is not intended as a function to login to on for login to the network would involve creating new n merely acquiring a handle to existing credentials.
	handle that will invo gss_init_sec_cor	<u>C_NO_NAME</u> , the call is interpreted as a request for a credential ke default behavior when passed to ntext(3GSS) (if <i>cred_usage</i> is GSS_C_INITIATE or GSS_C_BOTH) c_context(3GSS) (if <i>cred_usage</i> is GSS_C_ACCEPT or
	mechanisms requeste mechanisms can shar	<pre>ire_cred() returns a credential that is valid only for the ed by the <i>desired_mechs</i> argument. However, if multiple re a single credential element, the function returns all the eh the credential is valid in the <i>actual_mechs</i> argument.</pre>
	the GSS-API routines Accordingly, you ma means of gss_acqu Alternatively, you ma name produced when	() is intended to be used primarily by context acceptors, since obtain initiator credentials through the system login process. y not acquire GSS_C_INITIATE or GSS_C_BOTH credentials by ire_cred() for any name other than GSS_C_NO_NAME. ay acquire GSS_C_INITIATE or GSS_C_BOTH credentials for a n gss_inquire_cred(3GSS) is applied to a valid credential, or _context(3GSS) is applied to an active context.
	If credential acquisition is time-consuming for a mechanism, the mechanism may choose to delay the actual acquisition until the credential is required, for example, by gss_init_sec_context(3GSS) or by gss_accept_sec_context(3GSS). Such mechanism-specific implementations are, however, invisible to the calling application; thus a call of gss_inquire_cred(3GSS) immediately following the call of gss_acquire_cred() will return valid credential data and incur the overhead of a deferred credential acquisition.	
PARAMETERS	The parameter descri	ptions for gss_acquire_cred() follow:
	desired_name	The name of the principal for which a credential should be acquired.
	time_req	The number of seconds that credentials remain valid. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted lifetime

			gss_acquire_creu(5G55)
	desired_mechs		ing security mechanisms that may be used. SET may be used to obtain a default.
	cred_usage	flag is GSS_C_ACC accept security cre credentials will be	es how this credential should be used. If the CEPT, then credentials will be used only to dentials. GSS_C_INITIATE indicates that used only to initiate security credentials. If BOTH, then credentials may be used either to ecurity contexts.
	output_cred_handle	credential handle	ential handle. Resources associated with this must be released by the application after use _release_cred(3GSS)
	actual_mechs	associated with the application after u	isms for which the credential is valid. Storage e returned OID-set must be released by the se with a call to id_set(3GSS). Specify NULL if not required.
	time_rec		seconds for which the returned credentials Specify NULL if not required.
	minor_status	Mechanism specifi	ic status code.
<b>RETURN VALUES</b>	gss_acquire_cred	d() may return the	following status codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_BAD_MECH		An unavailable mechanism has been requested.
	GSS_S_BAD_NAMETY	(PE	The type contained within the <i>desired_name</i> parameter is not supported.
	GSS_S_BAD_NAME		The value supplied for <i>desired_name</i> parameter is ill formed.
	GSS_S_CREDENTIAI	LS_EXPIRED	The credentials could not be acquired because they have expired.
	GSS_S_NO_CRED		No credentials were found for the specified name.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5)	for descriptions of tl	he following attributes:

### gss\_acquire\_cred(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32–bit)
	SUNWgssx (64-bit)
MT-Level	Safe

## SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_init\_sec\_context(3GSS), gss\_inquire\_context(3GSS), gss\_inquire\_cred(3GSS), gss\_release\_cred(3GSS), gss\_release\_oid\_set(3GSS), attributes(5)

GSS-API Programming Guide

208 man pages section 3: Networking Library Functions • Last Revised 18 Apr 2000

NAME	gss_add_cred – add a credential-element to a credential
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>
	<pre>OM_uint32 gss_add_cred(OM_uint32 *minor_status, const gss_cred_id_t     input_cred_handle, const gss_name_t desired_name, const gss_OID     desired_mech, gss_cred_usage_t cred_usage, OM_uint32 initiator_time_req,     OM_uint32 acceptor_time_req, gss_cred_id_t *output_cred_handle,     gss_OID_set *actual_mechs, OM_uint32 *initiator_time_rec, OM_uint32     *acceptor_time_rec);</pre>
DESCRIPTION	The gss_add_cred() function adds a credential-element to a credential. The credential-element is identified by the name of the principal to which it refers. This routine is not intended as a function to login to the network; a function for login to the network would involve creating new mechanism-specific authentication data rather than merely acquiring a handle to existing data.
	If the value of <i>desired_name</i> is GSS_C_NO_NAME, the call is interpreted as a request to add a credential element that will invoke default behavior when passed to gss_init_sec_context(3GSS) (if the value of <i>cred_usage</i> is GSS_C_INITIATE or GSS_C_BOTH) or gss_accept_sec_context(3GSS) (if the value of <i>cred_usage</i> is GSS_C_ACCEPT or GSS_C_BOTH).
	The gss_add_cred() function is expected to be used primarily by context acceptors, since the GSS-API provides mechanism-specific ways to obtain GSS-API initiator credentials through the system login process. Consequently, the GSS-API therefore does not support acquiring GSS_C_INITIATE or GSS_C_BOTH credentials by means of gss_acquire_cred(3GSS) for any name other than GSS_C_NO_NAME, or from name produced by gss_inquire_cred(3GSS) applied to a valid credential or gss_inquire_context(3GSS) applied to an active context.
	If credential acquisition is time-consuming for a mechanism, the mechanism may choose to delay the actual acquisition until the credential is required, for example, by gss_init_sec_context(3GSS) or by gss_accept_sec_context(3GSS). Such mechanism-specific implementation decisions are, however, invisible to the calling application; thus a call to gss_inquire_cred(3GSS) immediately following the call of gss_add_cred() will return valid credential data as well as incur the overhead of deferred credential acquisition.
	The gss_add_cred() routine can be used either to compose a new credential that contains all credential-elements of the original in addition to the newly-acquired credential-element, or to add the new credential-element to an existing credential. If the value of the <i>output_cred_handle</i> parameter argument is NULL, the new credential-element will be added to the credential identified by <i>input_cred_handle</i> ; if a valid pointer is specified for the <i>output_cred_handle</i> parameter, a new credential handle will be created.

# gss\_add\_cred(3GSS)

s_add_cred(3G55)							
	If the value of <i>input_cred_handle</i> is GSS_C_NO_CREDENTIAL, gss_add_cred() will compose a credential and set the <i>output_cred_handle</i> parameter based on the default behavior. That is, the call will have the same effect as if the application had first made a call to gss_acquire_cred(3GSS) specifying the same usage and passing GSS_C_NO_NAME as the <i>desired_name</i> parameter to obtain an explicit credential handle to gss_add_cred(), and finally called gss_release_cred(3GSS) on the first credential handle.						
	If the value of the <i>input_cred_handle</i> parameter is GSS_C_NO_CREDENTIAL, you supply a non-NULL value for the <i>output_cred_handle</i> parameter.						
PARAMETERS	The parameter desc	riptions for gss_acquire_cred() follow:					
	minor_status	A mechanism specific status code.					
	input_cred_handle	The credential to which the credential-element will be added. If GSS_C_NO_CREDENTIAL is specified, the routine will compose the new credential based on default behavior. While the credential-handle is not modified by gss_add_cred(), if <i>output_credential_handle</i> is NULL, the underlying credential will be modified.					
	desired_name	Name of principal for which a credential should be acquired.					
	desired_mech	If the value of <i>desired_mech</i> is GSS_C_BOTH, the credential may be used either to initiate or accept security contexts. If the value of <i>desired_mech</i> is GSS_C_INITIATE, the credential will only be used to initiate security contexts. The credential will only be used to accept security contexts, if the value of <i>desired_mech</i> is GSS_C_ACCEPT.					
	initiator_time_req	The number of seconds that the credential may remain valid for initiating security contexts. This argument is ignored if the composed credentials are of type GSS_C_ACCEPT. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.					
	acceptor_time_req	Number of seconds that the credential may remain valid for accepting security contexts. This argument is ignored if the composed credentials are of type GSS_C_INITIATE. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.					
	output_cred_handle	The returned credential handle that contains the new credential-element and all the credential-elements from <i>input_cred_handle</i> . If a valid pointer to a gss_cred_id_t is supplied for this parameter, gss_add_cred() creates a new credential handle containing all credential-elements from <i>input_cred_handle</i> and the newly acquired credential-element; if					

			gss_add_cred(3GSS)	
		NULL is specified for this parameter, the newly acquired credential-element will be added to the credential identified <i>input_cred_handle</i> .		
		The resources associated with any credential handle returned by means of this parameter must be released by the application after use by a call to gss_release_cred(3GSS).		
	actual_mechs	valid. Storage for the application after use	mechanisms for which the new credential is e returned OID-set must be freed by the e by a call to gss_release_oid_set(3GSS). parameter is not required.	
	initiator_time_rec	The actual number of seconds for which the returned credentials will remain valid for initiating contexts using the specified mechanism. If a mechanism does not support expiration of credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if this parameter is not required The actual number of seconds for which the returned credentials will remain valid for accepting security contexts using the specified mechanism. If a mechanism does not support expiration of credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if this parameter is not required.		
	acceptor_time_rec			
<b>RETURN VALUES</b>	gss_acquire_cre	ed() may return the	following status codes:	
	GSS_S_COMPLETE		Successful completion.	
	GSS_S_BAD_MECH GSS_S_BAD_NAMETYPE GSS_S_BAD_NAME		An unavailable mechanism has been requested.	
			The type contained within the <i>desired_name</i> parameter is not supported.	
			The value supplied for <i>desired_name</i> parameter is ill formed.	
	GSS_S_DUPLICAT	E_ELEMENT	The credential already contains an element for the requested mechanism that has overlapping usage and validity period.	
	GSS_S_CREDENTIALS_EXPIRED GSS_S_NO_CRED		The credentials could not be added because they have expired.	
			No credentials were found for the specified name.	
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.	

## gss\_add\_cred(3GSS)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE	
Availability	SUNWgss (32–bit)	
	SUNWgssx (64-bit)	
MT-Level	Safe	

### SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_acquire\_cred(3GSS), gss\_init\_sec\_context(3GSS), gss\_inquire\_context(3GSS)gss\_inquire\_cred(3GSS), gss\_release\_cred(3GSS), gss\_release\_oid\_set(3GSS), attributes(5)

GSS-API Programming Guide

## gss\_add\_oid\_set\_member(3GSS)

NAME	gss_add_oid_set_member – add an object i	dentifier to an object identifier set			
SYNOPSIS	cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi>				
	OM_uint32 gss_add_oid_set_member gss_OID member_oid, gss_OID_se	—			
DESCRIPTION	The gss_add_oid_set_member() function adds an object identifier to an object identifier set. You should use this function in conjunction with gss_create_empty_oid_set(3GSS) when constructing a set of mechanism OIDs for input to gss_acquire_cred(3GSS). The <i>oid_set</i> parameter must refer to an OID-set created by GSS-API, that is, a set returned by gss_create_empty_oid_set(3GSS). The <i>oid_set</i> parameter must refer to an OID-set created by GSS-API, that is, a set returned by gss_create_empty_oid_set(3GSS). The <i>oid_set</i> parameter must refer to an OID-set creates a copy of the <i>member_oid</i> and inserts this copy into the set, expanding the storage allocated to the OID-set elements array, if necessary. The function may add the new member OID anywhere within the elements array, and the GSS-API verifies that the new <i>member_oid</i> is not already contained within the elements array. If the <i>member_oid</i> is already present, the <i>oid_set</i> should remain unchanged.				
PARAMETERS	The parameter descriptions for gss_add_	oid_set_member() follow:			
	<i>minor_status</i> A mechanism specific s	status code.			
	<i>member_oid</i> Object identifier to be o	copied into the set.			
	<i>oid_set</i> Set in which the object	identifier should be inserted.			
<b>RETURN VALUES</b>	Thegss_add_oid_set_member() funct	ion may return the following status codes:			
	GSS_S_COMPLETE Successful completion.				
	specific GSS status	chanism detected an error for which no code is defined. The mechanism-specific d by means of the <i>minor_status</i> parameter ndition.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	ATTRIBUTE TYPE	ATTRIBUTE VALUE			
	Availability	SUNWgss (32-bit)			
		SUNWgssx (64-bit)			
	MT-Level	Safe			
SEE ALSO	gss_acquire_cred(3GSS), gss_create GSS-API Programming Guide	e_empty_oid_set(3GSS),attributes(5)			

gss\_canonicalize\_name(3GSS)

NAME	gss_canonicalize_name – convert an internal name to a mechanism name				
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>				
	<pre>OM_uint32 gss_canonicalize_name(OM_uint32 *minor_status, const gss_name_t input_name, const gss_OID mech_type, gss_name_t *output_name);</pre>				
DESCRIPTION	The gss_canonicalize_name() function generates a canonical mechanism name from an arbitrary internal name. The mechanism name is the name that would be returned to a context acceptor on successful authentication of a context where the initiator used the <i>input_name</i> in a successful call to gss_acquire_cred(3GSS), specifying an OID set containing <i>mech_type</i> as its only member, followed by a call to gss_init_sec_context(3GSS), specifying <i>mech_type</i> as the authentication mechanism.				
PARAMETERS	The parameter des	criptions	s for gss_canon	icalize_name() follow:	
	minor_status	Mechar	nism-specific stat	rus code.	
	input_name	The nar	ne for which a c	anonical form is desired.	
	mech_type	The authentication mechanism for which the canonical form of the name is desired. The desired mechanism must be specified explicitly; no default is provided. The resultant canonical name. Storage associated with this name must be freed by the application after use with a call to gss_release_name(3GSS).			
	output_name				
<b>RETURN VALUES</b>	The gss_canonic	calize_	name() function	n may return the status codes:	
	GSS_S_COMPLETE	]	Successful com	pletion.	
	GSS_S_BAD_MECH	I	The identified	mechanism is not supported.	
	GSS_S_BAD_NAME	TYPE		nternal name contains no elements that ssed by the specified mechanism.	
	GSS_S_BAD_NAME	]	The provided i	internal name was ill-formed.	
	GSS_S_FAILURE		no specific GSS mechanism-spe	g mechanism detected an error for which S status code is defined. The ecific status code reported by means of the arameter details the error condition.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	ATTF	RIBUTE TYP	PE	ATTRIBUTE VALUE	
	Availability SUNWgss (32–bit)				

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32–bit)

214 man pages section 3: Networking Library Functions • Last Revised 18 Apr 2000

gss\_canonicalize\_name(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	SUNWgssx (64-bit)	
MT-Level	Safe	

### SEE ALSO

gss\_acquire\_cred(3GSS), gss\_init\_sec\_context(3GSS), gss\_release\_name(3GSS), attributes(5)

GSS-API Programming Guide

gss\_compare\_name(3GSS)

NAME	gss_compare_nam	ne – comp	oare two internal	-form names	
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>				
	<pre>OM_uint32 gss_compare_name(OM_uint32 *minor_status, const gss_name_t</pre>				ss_name_t
DESCRIPTION	The gss_compare_name() function allows an application to compare two internal-form names to determine whether they refer to the same entity.				)
	If either name presented to gss_compare_name() denotes an anonymous principal, the routines indicate that the two names do not refer to the same identity.				s principal,
PARAMETERS	The parameter des	scriptions	s for gss_compa	are_name() follow:	
	minor_status	Mechar	nism-specific sta	tus code.	
	name1	Interna	l-form name.		
	name2	Interna	l-form name.		
	name_equal	If non-zero, the names refer to same entity. If 0, the names refer to different entities. Strictly, the names are not known to refer to the same identity.			
<b>RETURN VALUES</b>	Thegss_compar	e_name(	) function may	return the following status code	s:
	GSS_S_COMPLETE Successful co		Successful con	npletion.	
	GSS_S_BAD_NAM	BAD_NAMETYPE The two name		es were of incomparable types.	
	GSS_S_BAD_NAM	_BAD_NAME One or both of		name1 or name2 was ill-formed.	
	GSS_S_FAILURE		no specific GSS mechanism-sp	g mechanism detected an error f 5 status code is defined. The ecific status code reported by me arameter details the error condit	eans of the
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	ATT	RIBUTE TY	PE	ATTRIBUTE VALUE	
	Availability			SUNWgss (32-bit)	
				SUNWgssx (64–bit)	
	MT-Level			Safe	
SEE ALSO	attributes(5) GSS-API Program	ming Gu	ide	·	

NAME	gss_context_time – determine how long a context will remain valid		
SYNOPSIS	cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	<pre>OM_uint32 gss_context_time(OM_uint32 *minor_status, gss_ctx_id_t</pre>		
DESCRIPTION	The gss_context_time() function determines the number of seconds for which the specified context will remain valid.		
PARAMETERS	The parameter des	scriptions for gss_conte	ext_time() are as follows:
	minor_status	A mechanism-specific s	tatus code.
	context_handle	A read-only value. Iden	tifies the context to be interrogated.
	<i>time_rec</i> Modifies the number of seconds that the context remains valid. If the context has already expired, returns zero.		
<b>RETURN VALUES</b>	The gss_contex	t_time() function retur	ns one of the following status codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_CONTEXT_EXPIRED		The context has already expired.
			The <i>context_handle</i> parameter did not dentify a valid context.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code s defined. The mechanism-specific status ode reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT Level	Safe

SEE ALSO
gss\_init\_sec\_context(3GSS), gss\_accept\_sec\_context(3GSS),
gss\_delete\_sec\_context(3GSS), gss\_process\_context\_token(3GSS),
gss\_inquire\_context(3GSS), gss\_wrap\_size\_limit(3GSS),
gss\_export\_sec\_context(3GSS), gss\_import\_sec\_context(3GSS),
attributes(5)

## gss\_create\_empty\_oid\_set(3GSS)

0 1 5			
NAME	gss_create_empty_ identifiers	oid_set – create an objec	t-identifier set containing no object
SYNOPSIS	<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi></pre>		
	OM_uint32 <b>gss</b> _ gss_OID_se		<pre>et(OM_uint32 *minor_status,</pre>
DESCRIPTION	containing no object the gss_add_oid	ct identifiers to which mo _set_member(3GSS) fu	nction creates an object-identifier set embers may be subsequently added using nction. These functions can be used to ers for input to gss_acquire_cred(3GSS).
PARAMETERS	The parameter des	criptions for gss_creat	<pre>ce_empty_oid_set() follow:</pre>
	minor_status	Mechanism-specific stat	tus code
	oid_set	gss_OID_set_desc o	set. The function will allocate the bject, which the application must free after release_oid_set(3GSS).
<b>RETURN VALUES</b>	The gss_create_ codes:	_empty_oid_set() fur	nction may return the following status
	GSS_S_COMPLETE	Successful completion	on
	GSS_S_FAILURE	specific GSS status c	chanism detected an error for which no code is defined. The mechanism-specific by means of the <i>minor_status</i> parameter dition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		following attributes:
	ATTR	IBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe
SEE ALSO		ed(3GSS), gss_add_oi d_set(3GSS), attribu ning Guide	
	I		

218 man pages section 3: Networking Library Functions • Last Revised Apr 18 2000

## gss\_delete\_sec\_context(3GSS)

NAME	gss_delete_sec_cor	ntext – delete a GSS-API s	security context
SYNOPSIS	cc -flag <i>file</i> #include <gssapi< th=""><th>lgss [ library /gssapi.h&gt;</th><th>. ]</th></gssapi<>	lgss [ library /gssapi.h>	. ]
			<pre>(OM_uint32 *minor_status, s_buffer_t output_token);</pre>
DESCRIPTION	Use the gss_delete_sec_context() function to delete a security context. The gss_delete_sec_context() function will delete the local data structures associated with the specified security context. You may not obtain further security services that use the context specified by <i>context_handle</i> .		
	will delete any hal	f-built security contexts t	contexts, gss_delete_sec_context() hat result from incomplete sequences of nd gss_accept_sec_context(3GSS).
	The Solaris implementation of the GSS-API retains the <i>output_token</i> parameter for compatibility with version 1 of the GSS-API. Both peer applications should invoke gss_delete_sec_context(), passing the value GSS_C_NO_BUFFER to the <i>output_token</i> parameter; this indicates that no token is required. If the application passes a valid buffer to gss_delete_sec_context(), it will return a zero-length token, indicating that no token should be transferred by the application.		
PARAMETERS	The parameter descriptions for gss_delete_sec_context() follow:		
	minor_status	A mechanism specific s	tatus code.
	context_handle		ing specific context to delete. After deleting I will set <i>context_handle</i> to
	output_token	A token to be sent to readelete the context.	mote applications that instructs them to
<b>RETURN VALUES</b>	gss_delete_sec	c_context() may return	n the following status codes:
	GSS_S_COMPLETE	E Successful c	ompletion.
	GSS_S_NO_CONTE	EXT No valid con	ntext was supplied.
	GSS_S_FAILURE	no specific C mechanism-	ing mechanism detected an error for which GSS status code is defined. The specific status code reported by means of <i>atus</i> parameter details the error condition.
ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATTE	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)

Networking Library Functions 219

gss\_delete\_sec\_context(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
	SUNWgssx (64-bit)
MT-Level	Safe

SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_init\_sec\_context(3GSS), attributes(5)

NAME	gss_display_name – convert internal-form name to text		
SYNOPSIS	<b>cc</b> [ <i>flag</i> ] <i>file</i> ] #include <gssapi gs<="" th=""><th></th><th></th></gssapi>		
	<pre>OM_uint32 gss_display_name(OM_uint32 *minor_status, const gss_name_t</pre>		
DESCRIPTION	<b>DESCRIPTION</b> The gss_display_name() function allows an application to obtain a textua representation of an opaque internal-form name for display purposes.		
	value GSS_C_NT_AN	ONYMOUS as the <i>outpu</i>	ipal, the GSS-API returns the gss_OID <i>t_name_type</i> , and a textual name that is red printable names in <i>output_name_buffer</i> .
		ie name-type, the $GSS$	import_name(3GSS), specifying -API returns GSS_C_NO_OID by means of
PARAMETERS	The parameter descri	ptions for gss_disp1	lay_name() follow:
	minor_status	Mechanism-specific	status code.
	input_name	Name in internal for	m.
	output_name_buffer		tual name string. The application must free rith this name after use with a call to fer(3GSS).
	output_name_type	pointer into static static the caller. In particul	rned name. The returned gss_OID will be a orage and should be treated as read-only by lar, the application should not attempt to if this parameter is not required.
<b>RETURN VALUES</b>	The gss_display_r	name() function may	return the following status codes:
	GSS_S_COMPLETE	Successful completi	on.
	GSS_S_BAD_NAME	The <i>input_name</i> was	s ill-formed.
	GSS_S_FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specifi status code reported by means of the <i>minor_status</i> parame details the error condition.		code is defined. The mechanism-specific l by means of the <i>minor_status</i> parameter
ATTRIBUTES	See attributes(5) f	or descriptions of the	following attributes:
	ATTRIBU	JTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			·

Networking Library Functions 221

## gss\_display\_name(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** gss\_import\_name(3GSS), gss\_release\_buffer(3GSS), attributes(5)

# gss\_display\_status(3GSS)

NAME	gss_display_status – convert a GSS-API status code to text			
SYNOPSIS	cc -flag <i>fil</i> #include <gssap:< th=""><th>elgss [ library ] i/gssapi.h&gt;</th></gssap:<>	elgss [ library ] i/gssapi.h>		
	<pre>OM_uint32 gss_display_status(OM_uint32 *minor_status, OM_uint32 status value, int status type, const gss_OID mech_type, OM_uint32 *message_context, gss_buffer_t status string);</pre>			
DESCRIPTION	The gss_display_status() function enables an application to obtain a textual representation of a GSS-API status code for display to the user or for logging purposes. Because some status values may indicate multiple conditions, applications may need to call gss_display_status() multiple times, with each call generating a single text string.			
	information on er message_context pa call, and gss_dis	<pre>ext parameter is used by gss_acquire_cred() to store state ror messages that are extracted from a given status_value. The arameter must be initialized to 0 by the application prior to the first splay_status() will return a non-zero value in this parameter if messages to extract.</pre>		
	The <i>message_context</i> parameter contains all state information required by gss_display_status() to extract further messages from the <i>status_value</i> . If a non-zero value is returned in this parameter, the application is not required to call gss_display_status() again unless subsequent messages are desired.			
PARAMETERS	The parameter de	scriptions for gss_display_status() follow:		
	minor_status	Status code returned by the underlying mechanism.		
	status_value	Status value to be converted.		
	status_type	If the value is GSS_C_GSS_CODE, <i>status_value</i> is a GSS-API status code. If the value is GSS_C_MECH_CODE, then <i>status_value</i> is a mechanism status code.		
	mech_type	Underlying mechanism that is used to interpret a minor status value. Supply GSS_C_NO_OID to obtain the system default.		
	message_context	Should be initialized to zero prior to the first call. On return from gss_display_status(), a non-zero <i>status_value</i> parameter indicates that additional messages may be extracted from the status code by means of subsequent calls to gss_display_status(), passing the same <i>status_value</i> , <i>status_type</i> , <i>mech_type</i> , and <i>message_context</i> parameters.		
	status_string	Textual representation of the <i>status_value</i> . Storage associated with this parameter must be freed by the application after use with a call to gss_release_buffer(3GSS).		
RETURN VALUES	The gss_displa	$y_status()$ function may return the following status codes:		

## gss\_display\_status(3GSS)

	GSS_S_COMPLETE	Successful compl	letion.
	GSS_S_BAD_MECH	Indicates that tra mechanism type	nslation in accordance with an unsupported was requested.
	GSS_S_BAD_STATUS	The status value was not recognized, or the status type was neither GSS_C_GSS_CODE nor GSS_C_MECH_CODE.	
	GSS_S_FAILURE	specific GSS statu	nechanism detected an error for which no us code is defined. The mechanism-specific red by means of the <i>minor_status</i> parameter condition.
ATTRIBUTES	See attributes(5) for	descriptions of the	following attributes:
	ATTRIBUTE	ТҮРЕ	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe
	GSS-API Programming	Guide	

## gss\_duplicate\_name(3GSS)

NAME	gss_duplicate_name – create a copy of an internal name			
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>			
			<pre>plicate_name(OM_ rc_name, gss_name_</pre>	uint32 * <i>minor_status</i> , const _t * <i>dest_name</i> );
DESCRIPTION	The gss_duplicate_name() function creates an exact duplicate of the existing internal name <i>src_name</i> . The new <i>dest_name</i> will be independent of the <i>src_name</i> . The <i>src_name</i> and <i>dest_name</i> must both be released, and the release of one does not affect the validity of the other.			
PARAMETERS	The parameter	descri	ptions for gss_dupli	cate_name() follow:
	minor_status	A me	chanism-specific statu	ıs code.
	src_name	Inter	nal name to be duplica	ated.
	dest_name	must		<i>ame</i> . Storage associated with this name cation after use with a call to S).
<b>RETURN VALUES</b>	The gss_dupl	icate		ay return the following status codes:
	GSS_S_COMPL	ETE	Successful completion	on.
	GSS_S_BAD_N	AME	The <i>src_name</i> param	eter was ill-formed.
	GSS_S_FAILURE The underlying mechanism detected an error for which specific GSS status code is defined. The mechanism-spec status code reported by means of the <i>minor_status</i> param details the error condition.		ode is defined. The mechanism-specific l by means of the <i>minor_status</i> parameter	
ATTRIBUTES	See attribute	es(5) f	or descriptions of the	following attributes:
		ATTRIBL	ІТЕ ТҮРЕ	ATTRIBUTE VALUE
	Availability			SUNWgss (32-bit)
SUNWgssx (64-b		SUNWgssx (64-bit)		
	MT-Level Safe		Safe	
SEE ALSO	gss_release GSS-API Progra	_	(3GSS), attributes ng Guide	(5)

Networking Library Functions 225

#### gss\_export\_name(3GSS)

NAME	gss_export_name – conve	ert a mechanism n	ame to export form	
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>			
	OM_uint32 <b>gss_expor</b> <i>input_name</i> , gss_k		t32 * <i>minor_status</i> , const gss_name_t d_name);	
DESCRIPTION	The gss_export_name() function allows a GSS-API internal name to be converted into a mechanism-specific name. The function produces a canonical contiguous string representation of a mechanism name, suitable for direct comparison, with memcmp(3C), or for use in authorization functions, matching entries in an access-control list. The <i>input_name</i> parameter must specify a valid mechanism name, that is, an internal name generated by gss_accept_sec_context(3GSS) or by gss_canonicalize_name(3GSS).			
PARAMETERS	The parameter description	ns for gss_expo	rt_name() follow:	
	<i>minor_status</i> A mech	anism-specific sta	tus code.	
	<i>input_name</i> The me	chanism name to	be exported.	
	associat		s string form of <i>input_name</i> . Storage g must freed by the application after use ffer(3GSS).	
<b>RETURN VALUES</b>	Thegss_export_name	() function may r	eturn the following status codes:	
	GSS_S_COMPLETE	Successful comp	pletion.	
	GSS_S_NAME_NOT_MN	The provided ir	nternal name was not a mechanism name.	
	GSS_S_FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.			
ATTRIBUTES	See attributes(5) for d	lescriptions of the	following attributes:	
	ATTRIBUTE T	YPE	ATTRIBUTE VALUE	
	Availability		SUNWgss (32-bit)	
	SUNWgssx (64–bit)			
MT-Level Safe				
SEE ALSO	gss_accept_sec_cont gss_release_buffer( GSS-API Programming Gu	3GSS)memcmp(3C	canonicalize_name(3GSS), ),attributes(5)	
	J			

226 man pages section 3: Networking Library Functions • Last Revised 27 Mar 2000

## gss\_export\_sec\_context(3GSS)

		gos_export_sec_context(0000)	
NAME			
SYNOPSIS	<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi></pre>		
		<pre>ec_context(OM_uint32 *minor_status, kt_handle, gss_buffer_t interprocess_token);</pre>	
DESCRIPTION	transfer to another process wi	<pre>s_export_sec_context() function generates an interprocess token for r to another process within an end system.gss_export_sec_context() and mport_sec_context() allow a security context to be transferred between ses on a single machine.</pre>	
	The gss_export_sec_context() function supports the sharing of work between multiple processes. This routine is typically used by the context-acceptor, in an application where a single process receives incoming connection requests and accepts security contexts over them, then passes the established context to one or more other processes for message exchange.gss_export_sec_context() deactivates the security context for the calling process and creates an interprocess token which, when passed to gss_import_sec_context() in another process, reactivates the context in the second process. Only a single instantiation of a given context can be active at any one time; a subsequent attempt by a context exporter to access the exported security context will fail.		
	The interprocess token may contain security-sensitive information, for example cryptographic keys. While mechanisms are encouraged to either avoid placing such sensitive information within interprocess tokens or to encrypt the token before returning it to the application, in a typical object-library GSS-API implementation, this might not be possible. Thus, the application must take care to protect the interprocess token and ensure that any process to which the token is transferred is trustworthy. If creation of the interprocess token is successful, the GSS-API deallocates all process-wide resources associated with the security context and sets the context_handle to GSS_C_NO_CONTEXT. In the event of an error that makes it impossible to complete the export of the security context, the function does not return an interprocess token and leaves the security context referenced by the <i>context_handle</i> parameter untouched.		
	Sun's implementation of gss_export_sec_context() does not encrypt the interprocess token. The interprocess token is serialized before it is transferred to another process.		
PARAMETERS	The parameter descriptions for	orgss_export_sec_context() are as follows:	
	minor_status	A mechanism-specific status code.	
	context_handle	Context handle identifying the context to transfer.	
	<i>interprocess_token</i> Token to be transferred to target process. Storage associated with this token must be freed by the application after use with a call to gss release buffer(3GSS).		

gss\_export\_sec\_context(3GSS)

<b>RETURN VALUES</b>	gss_export_sec_context() returns one of the following status codes:		
	GSS_S_COMPLETE	Successful completion.	
	GSS_S_CONTEXT_EXPIRED The context has expired.		
	GSS_S_NO_CONTEXT	The context was invalid.	
	GSS_S_UNAVAILABLE	The operation is not supported.	
	GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.	

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32–bit)
	SUNWgssx (64-bit)
MT Level	Safe

SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_import\_sec\_context(3GSS), gss\_init\_sec\_context(3GSS), gss\_release\_buffer(3GSS), attributes(5)

			00()
NAME	gss_get_mic – ca	lculate a cryptographic i	nessage
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	context_han		2 * <i>minor_status</i> , const gss_ctx_id_t eq, const gss_buffer_t <i>message_buffer</i> ,
DESCRIPTION	message, and pla <i>qop_req</i> paramete	aces the MIC in a token f	a cryptographic MIC for the supplied for transfer to the peer application. The en several cryptographic algorithms, if
		) to provide secure fram	ay wish to use tokens emitted by ing, the GSS-API allows MICs to be derived
PARAMETERS	The parameter de	escriptions for gss_get	_mic() follow:
	minor_status	The status code returned	ed by the underlying mechanism.
	context_handle	Identifies the context o	n which the message will be sent.
	qop_req	on portability grounds offered by the chosen r specifying GSS_C_QOF	quality of protection. Callers are encouraged, , to accept the default quality of protection nechanism, which may be requested by 
	<i>message_buffer</i> The message to be protected.		
	msg_token		ne token. Storage associated with this message oplication after use with a call to er(3GSS).
<b>RETURN VALUES</b>	gss_get_mic()	) may return the followi	ng status codes:
	GSS_S_COMPLE	ГЕ	Successful completion.
	GSS_S_CONTEXT	I_EXPIRED	The context has already expired.
	GSS_S_NO_CON	TEXT	The <i>context_handle</i> parameter did not identify a valid context.
	GSS_S_BAD_QOI	P	The specified QOP is not supported by the mechanism.
	GSS_S_FAILURI	Ξ	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
			N

#### gss\_get\_mic(3GSS)

ATTRIBUTES

 $\mathbf{S} \mid$  See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32–bit)
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** gss\_release\_buffer(3GSS), gss\_wrap(3GSS), attributes(5)

			0
NAME	gss_import_name –	conver	t a contiguous string name to GSS_API internal format
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>		
		_t inpu	_ <b>name</b> (OM_uint32 * <i>minor_status</i> , const tt_name_buffer, const gss_OID input_name_type, t_name);
DESCRIPTION	The gss_import_name() function converts a contiguous string name to internal form. In general, the internal name returned by means of the <i>output_name</i> parameter will not be a mechanism name; the exception to this is if the <i>input_name_type</i> indicates that the contiguous string provided by means of the <i>input_name_buffer</i> parameter is of type GSS_C_NT_EXPORT_NAME, in which case, the returned internal name will be a mechanism name for the mechanism that exported the name.		
PARAMETERS	The parameter descr	riptions	s for gss_import_name() follow:
	minor_status	Status	s code returned by the underlying mechanism.
	input_name_buffer	impo: argun	uss_buffer_desc structure containing the name to be rted. The application must allocate this explicitly. This nent must be deallocated with release_buffer(3GSS) when the application is done it.
	input_name_type	Agss in.	s_OID that specifies the format that the <i>input_name_buffer</i> is
	output_name	The g	ss_name_t structure to receive the name.
<b>RETURN VALUES</b>	Thegss_import_r	name()	function may return the following status codes:
	GSS_S_COMPLETE		The gss_import_name() function completed successfully.
	GSS_S_BAD_NAMET	YPE	The <i>input_name_type</i> was unrecognized.
	GSS_S_BAD_NAME		The <i>input_name</i> parameter could not be interpreted as a name of the specified type.
	GSS_S_BAD_MECH		The <i>input_name_type</i> was GSS_C_NT_EXPORT_NAME, but the mechanism contained within the <i>input_name</i> is not supported.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5)	for des	scriptions of the following attributes:

#### gss\_import\_name(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** gss\_release\_buffer(3GSS), attributes(5)

	<b>č</b>
gss_import_sec_context - imp	ort security context established by another process
<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""> OM_uint32 gss_import_sec_context(OM_uint32 *minor_status, const     gss_buffer_t interprocess_token, gss_ctx_id_t *context_handle);</gssapi></pre>	
The parameter descriptions for gss_import_sec_context() are as follows:	
minor_status	A mechanism-specific status code.
interprocess_token	Token received from exporting process.
context_handle	Context handle of newly reactivated context. Resource associated with this context handle must be released by the application after use with a call to gss_delete_sec_context(3GSS).
gss_import_sec_context	() returns one of the following status codes:
GSS_S_COMPLETE	Successful completion.
GSS_S_NO_CONTEXT	The token did not contain a valid context reference.
GSS_S_DEFECTIVE_TOKEN	The token was invalid.
GSS_S_UNAVAILABLE	The operation is unavailable.
GSS_S_UNAUTHORIZED	Local policy prevents the import of this context by the current process.
GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
See attributes(5) for descriptions of the following attributes:	
ATTRIBUTE TYPE	ATTRIBUTE VALUE
	<pre>cc -flag filelgss [ #include <gssapi gssapi.h=""> OM_uint32 gss_import_sec gss_buffer_t interprot The gss_import_sec_cont context established by another only once. See gss_export_i The parameter descriptions fo minor_status interprocess_token context_handle gss_import_sec_context GSS_S_COMPLETE GSS_S_NO_CONTEXT GSS_S_DEFECTIVE_TOKEN GSS_S_UNAVAILABLE GSS_S_UNAVAILABLE GSS_S_FAILURE See attributes(5) for description See attributes(5) for description Set attributes(5</gssapi></pre>

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64–bit)
MT Level	Safe

gss\_import\_sec\_context(3GSS)

```
SEE ALSO gss_accept_sec_context(3GSS), gss_context_time(3GSS),
gss_delete_sec_context(3GSS), gss_export_sec_content(3GSS),
gss_init_sec_context(3GSS), gss_inquire_context(3GSS),
gss_process_context_token(3GSS), gss_wrap_size_limit(3GSS),
attributes(5)
```

			0 = -
NAME	gss_indicate_mecl	ns – determine available	security mechanisms
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	OM_uint32 <b>gss</b> * <i>mech_set</i> );	_indicate_mechs(OM_	_uint32 * <i>minor_status</i> , gss_OID_set
DESCRIPTION	The gss_indicate_mechs() function enables an application to determine available underlying security mechanisms.		
PARAMETERS	The parameter descriptions for gss_indicate_mechs() follow:		
	minor_status A	mechanism-specific stat	rus code.
	b	e a dynamically-allocated	sms. The returned gss_OID_set value will d OID set that should be released by the to gss_release_oid_set(3GSS).
<b>RETURN VALUES</b>	The gss_indica	te_mechs() function m	nay return the following status codes:
	GSS_S_COMPLET	E Successful complet	ion.
	GSS_S_FAILURE	specific GSS status	chanism detected an error for which no code is defined. The mechanism-specific d by means of the <i>minor_status</i> parameter ndition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe
SEE ALSO	gss_release_o GSS-API Program	id_set(3GSS), attribu ming Guide	utes(5)

NAME	gss_init_sec_context – initiate a GSS-API security context with a peer application		
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	<pre>OM_uint32 gss_init_sec_context(OM_uint32 *minor_status, const gss_cred_id_t initiator_cred_handle, gss_ctx_id_t *context_handle, const gss_name_t *target_name, const gss_OID mech_type, OM_uint32 req_flags, OM_uint32 time_req, const gss_channel_bindings_t input_chan_bindings, const gss_buffer_t input_token, gss_OID *actual_mech_type, gss_buffer_t output_token, OM_uint32 *ret_flags, OM_uint32 *time_rec);</pre>		
DESCRIPTION	The gss_init_sec_context() function initiates the establishment of a security context between the application and a remote peer. Initially, the <i>input_token</i> paramet should be specified either as GSS_C_NO_BUFFER, or as a pointer to a gss_buffer_desc object with a length field that contains a zero value. The routi may return a <i>output_token</i> , which should be transferred to the peer application, whice will present it to gss_accept_sec_context(3GSS). If no token need be sent, gss_init_sec_context() will indicate this by setting the length field of the <i>output_token</i> argument to zero. To complete context establishment, one or more reply tokens may be required from the peer application; if so, gss_init_sec_context() will return a status code that contains the supplementary information bit GSS_S_CONTINUE_NEEDED. In this case, make another call to gss_init_sec_context() when the reply token is received from the peer application and pass the reply token to gss_init_sec_context() by means of th <i>input_token</i> parameter.		
	Construct portable applications to use the token length and return status to determine whether to send or wait for a token.		
	Whenever the routine returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established, and the following restrictions apply to the output parameters:		
	The value returned by means of the <i>time_rec</i> parameter is undefined. Unless the accompanying <i>ret_flags</i> parameter contains the bit GSS_C_PROT_READY_FLAG, which indicates that per-message services may be applied in advance of a successful completion status, the value returned by means of the <i>actual_mech_type</i> parameter is undefined until the routine returns a major status value of GSS_S_COMPLETE.		
	The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned by the <i>ret_flags</i> parameter contain values that will be valid if context establishment succeeds. For example, if the application requests a service such as delegation or anonymous authentication by means of the <i>req_flags</i> argument, and the service is unavailable from the underlying mechanism, gss_init_sec_context() generates a token that will not provide the service, and it indicate by means of the <i>ret_flags</i> argument		

	establishment by ca without the service	l not be supported. The application may choose to abort context alling gss_delete_sec_context(3GSS) if it cannot continue e, or if the service was merely desired but not mandatory, it may and continue context establishment.		
	The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within ret_flags indicate the actual state at the time gss_init_sec_context() returns, whether or not the context is fully established.			
	caller, for example, However, applicati in Version 1 of the per-message servic according to the GS	he GSS_C_PROT_READY_FLAG in the final <i>ret_flags</i> returned to a when accompanied by a GSS_S_COMPLETE status code. ons should not rely on this behavior, as the flag was not defined GSS-API. Instead, applications should determine what es are available after a successful context establishment SS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.		
	<ul> <li>All other bits withit</li> </ul>	n the <i>ret_flags</i> argument are set to zero.		
	context object; it leaves GSS_C_NO_CONTEXT	<pre>s_init_sec_context() fails, the GSS-API does not create a s the value of the context_handle parameter set to to indicate this. In the event of failure on a subsequent call, the curity context untouched for the application to delete using ontext(3GSS).</pre>		
	GSS_S_DUPLICATE_1 always return them in requirement was not p	shment, the informational status bits GSS_S_OLD_TOKEN and TOKEN indicate fatal errors, and GSS-API mechanisms should association with a status code of GSS_S_FAILURE. This pairing part of Version 1 of the GSS-API specification, so applications rsion 1 implementations must special-case these codes.		
PARAMETERS	The parameter descrip	tions for gss_init_sec_context() follow:		
	minor_status	A mechanism specific status code.		
	initiator_cred_handle	The handle for the credentials claimed. Supply GSS_C_NO_CREDENTIAL to act as a default initiator principal. If no default initiator is defined, the function returns GSS_S_NO_CRED.		
	context_handle	The context handle for a new context. Supply the value GSS_C_NO_CONTEXT for the first call, and use the value returned in any continuation calls. The resources associated with <i>context_handle</i> must be released by the application after use by a call to gss_delete_sec_context(3GSS).		
	target_name	The name of the target.		
	mech_type	The object ID of the desired mechanism. To obtain a specific default, supply the value GSS_C_NO_ID.		
	req_flags	Contains independent flags, each of which will request that the context support a specific service option. A symbolic name		

	is provided for each flag. Logically-OR the symbolic name to the corresponding required flag to form the bit-mask value. <i>req_flags</i> may contain one of the following values:
	GSS_C_DELEG_FLAG If true, delegate credentials to a remote peer. Do not delegate the credentials if the value is false.
	GSS_C_MUTUAL_FLAG If true, request that the peer authenticate itself. If false, authenticate to the remote peer only.
	GSS_C_REPLAY_FLAG If true, enable replay detection for messages protected with gss_wrap(3GSS) or gss_get_mic(3GSS). Do not attempt to detect replayed messages if false.
	GSS_C_SEQUENCE_FLAG If true, enable detection of out-of-sequence protected messages. Do not attempt to detect out-of-sequence messages if false.
	GSS_C_CONF_FLAG If true, request that confidential service be made available by means of gss_wrap(3GSS). If false, no per-message confidential service is required.
	GSS_C_INTEG_FLAG If true, request that integrity service be made available by means of gss_wrap(3GSS) or gss_get_mic(3GSS). If false, no per-message integrity service is required.
	GSS_C_ANON_FLAG If true, do not reveal the initiator's identify to the acceptor. If false, authenticate normally.
time_req	The number of seconds for which the context will remain valid. Supply a zero value to <i>time_req</i> to request a default validity period.
input_chan_bindings	Optional application-specified bindings. Allows application to securely bind channel identification information to the security context. Set to GSS_C_NO_CHANNEL_BINDINGS if you do not want to use channel bindings.
input_token	Token received from the peer application. On the initial call, supply GSS_C_NO_BUFFER or a pointer to a buffer containing the value GSS_C_EMPTY_BUFFER.
actual_mech_type	The actual mechanism used. The OID returned by means of this parameter will be pointer to static storage that should be treated as read-only. The application should not attempt to free

	it. To obtain a specific default, supply the value GSS_C_NO_ID. Specify NULL if the parameter is not required.
output_token	The token to send to the peer application. If the length field of the returned buffer is zero, no token need be sent to the peer application. After use storage associated with this buffer must be freed by the application by a call to gss_release_buffer(3GSS).
ret_flags	Contains various independent flags, each of which indicates that the context supports a specific service option. If not needed, specify NULL. Test the returned bit-mask <i>ret_flags</i> value against its symbolic name to determine if the given option is supported by the context. <i>ret_flags</i> may contain one of the following values:
	GSS_C_DELEG_FLAG If true, credentials were delegated to the remote peer. If false, no credentials were delegated.
	GSS_C_MUTUAL_FLAG If true, the remote peer authenticated itself. If false, the remote peer did not authenticate itself.
	GSS_C_REPLY_FLAG If true, replay of protected messages will be detected. If false, replayed messages will not be detected.
	GSS_C_SEQUENCE_FLAG If true, out of sequence protected messages will be detected. If false, they will not be detected.
	GSS_C_CONF_FLAG If true, confidential service may be invoked by calling the gss_wrap() routine. If false, no confidentiality service is available by means of gss_wrap(3GSS).gss_wrap() will provide message encapsulation, data-origin authentication and integrity services only.
	GSS_C_INTEG_FLAG If true, integrity service may be invoked by calling either the gss_wrap(3GSS) or gss_get_mic(3GSS) routine. If false, per-message integrity service is not available.
	GSS_C_ANON_FLAG If true, the initiator's identity has not been revealed; it will not be revealed if any emitted token is passed to the acceptor. If false, the initiator has been or will be authenticated normally.

		GSS_C_CONF_ if the accompa GSS_S_COMPI the protection	tection services specified by the states of FLAG and GSS_C_INTEG_FLAG are available nying major status return value is either LETE or GSS_S_CONTINUE_NEEDED. If false, services are available only if the major status return value is LETE.
		If true, the rest other processe gss_export_	ultant security context may be transferred to s by means of a call to _sec_context(3GSS). If false, the security be transferred.
	time_rec		conds for which the context will remain LL if the parameter is not required.
<b>RETURN VALUES</b>	gss_init_sec_cont	ext() may return	the following status codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_CONTINUE_NE	EEDED	A token from the peer application is required to complete the context, and gss_init_sec_context() must be called again with that token.
	GSS_S_DEFECTIVE_T	COKEN	Consistency checks performed on the <i>input_token</i> failed.
	GSS_S_DEFECTIVE_C	CREDENTIAL	Consistency checks performed on the credential failed.
	GSS_S_NO_CRED		The supplied credentials are not valid for context acceptance, or the credential handle does not reference any credentials.
	GSS_S_CREDENTIALS	S_EXPIRED	The referenced credentials have expired.
	GSS_S_BAD_BINDING	3S	The <i>input_token</i> contains different channel bindings than those specified by means of the <i>input_chan_bindings</i> parameter.
	GSS_S_BAD_SIG		The <i>input_token</i> contains an invalid MIC or a MIC that cannot be verified.
	GSS_S_OLD_TOKEN		The <i>input_token</i> is too old. This is a fatal error while establishing context.
	GSS_S_DUPLICATE_T	COKEN	The <i>input_token</i> is valid, but it is a duplicate of a token already processed. This is a fatal error while establishing context.

		gss_init_sec_context(3GSS)
	GSS_S_NO_CONTEXT	The supplied context handle does not refer to a valid context.
	GSS_S_BAD_NAMETYPE	The provided <i>target_name</i> parameter contains an invalid or unsupported <i>name</i> type.
	GSS_S_BAD_NAME	The supplied <i>target_name</i> parameter is ill-formed.
	GSS_S_BAD_MECH	The token received specifies a mechanism that is not supported by the implementation or the provided credential.
	GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
EXAMPLES	<b>EXAMPLE 1</b> Invoking gss_init_sec_co	ntext() Within a Loop
	A typical portable caller should alway loop:	sinvokegss_init_sec_context() within a
	<pre>int context_established = 0; gss_ctx_id_t context_hdl = GSS_C_NO_</pre>	CONTEXT;
	input_token->length = 0;	
	<pre>while (!context_established) {     mai stat = gag init gag gontext (sm </pre>	Nin stat
	<pre>maj_stat = gss_init_sec_context(&amp;m</pre>	red_hdl,
		context_hdl, arget name,
		esired_mech,
	-	sired_services, sired_time,
	in	put_bindings,
		uput_token, uctual_mech,
		ntput_token, Actual services,
		<pre>ictual_services, ictual_time);</pre>
	<pre>if (GSS_ERROR(maj_stat)) {     report error(maj stat, min stat)</pre>	
	};	,
	<pre>if (output_token-&gt;length != 0) {    send_token_to_peer(output_token)    gss_release_buffer(&amp;min_stat, ou</pre>	
	<pre>}; if (GSS_ERROR(maj_stat)) {</pre>	_
	if (context_hdl != GSS_C_NO_CONT	'EXT)

```
EXAMPLE 1 Invoking gss_init_sec_context() Within a Loop
                                                                            (Continued)
                     gss_delete_sec_context(&min_stat,
                                           &context hdl,
                                            GSS_C_NO_BUFFER);
                   break;
                  };
                 if (maj_stat & GSS_S_CONTINUE_NEEDED) {
                   receive_token_from_peer(input_token);
                  } else {
                   context_established = 1;
                  };
               };
ATTRIBUTES
               See attributes(5) for descriptions of the following attributes:
                             ATTRIBUTE TYPE
                                                                    ATTRIBUTE VALUE
                Availability
                                                        SUNWgss (32-bit)
                                                        SUNWgssx (64-bit)
                MT-Level
                                                        Safe
   SEE ALSO
               gss delete sec context(3GSS), gss export sec context(3GSS),
                gss_get_mic(3GSS), gss_wrap(3GSS), attributes(5)
                GSS-API Programming Guide
```

		0 - 1 - ( )	
NAME	gss_inquire_context – obtain information about a security context		
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	gss_ctx_ic *targ_name,	<pre>inquire_context(OM_uint32 *minor_status, const d_t context_handle, gss_name_t *src_name, gss_name_t OM_uint32 *lifetime_rec, gss_OID *mech_type, OM_uint32 int *locally_initiated, int *open);</pre>	
DESCRIPTION	context. The caller	e_context() function obtains information about a security must already have obtained a handle that refers to the context, ext need not be fully established.	
PARAMETERS	The parameter des	scriptions for gss_inquire_context() are as follows:	
	minor_status	A mechanism-specific status code.	
	context_handle	A handle that refers to the security context.	
	src_name	The name of the context initiator. If the context was established using anonymous authentication, and if the application invoking gss_inquire_context() is the context acceptor, an anonymous name is returned. Storage associated with this name must be freed by the application after use with a call to gss_release_name(). Specify NULL if the parameter is not required.	
	targ_name	The name of the context acceptor. Storage associated with this name must be freed by the application after use with a call to gss_release_name(). If the context acceptor did not authenticate itself, and if the initiator did not specify a target name in its call to gss_init_sec_context(), the value GSS_C_NO_NAME is returned. Specify NULL if the parameter is not required.	
	lifetime_rec	The number of seconds for which the context will remain valid. If the context has expired, this parameter will be set to zero. Specify NULL if the parameter is not required.	
	mech_type	The security mechanism providing the context. The returned OID is a pointer to static storage that should be treated as read-only by the application; in particular, the application should not attempt to free it. Specify NULL if the parameter is not required.	
	ctx_flags	Contains various independent flags, each of which indicates that the context supports (or is expected to support, if ctx_open is false) a specific service option. If not needed, specify NULL. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically ANDed with the ret_flags value to test whether a given option is supported by the context. The flags are:	

## gss\_inquire\_context(3GSS)

	GSS_C_DELEG_FLAG If true, credentials were delegated from the initiator to the acceptor. If false, no credentials were delegated.
	GSS_C_MUTUAL_FLAG If true, the acceptor was authenticated to the initiator. If false, the acceptor did not authenticate itself.
	GSS_C_REPLAY_FLAG If true, the replay of protected messages will be detected. If false, replayed messages will not be detected.
	GSS_C_SEQUENCE_FLAG If true, out-of-sequence protected messages will be detected. If false, out-of-sequence messages will not be detected.
	GSS_C_CONF_FLAG If true, confidential service may be invoked by calling the gss_wrap(3GSS) routine. If false, no confidential service is available through gss_wrap().gss_wrap() provides message encapsulation, data-origin authentication, and integrity services only.
	GSS_C_INTEG_FLAG If true, integrity service can be invoked by calling either the gss_get_mic() or the gss_wrap() routine. If false, per-message integrity service is unavailable.
	GSS_C_ANON_FLAG If true, the initiator's identity is not revealed to the acceptor. The <i>src_name</i> parameter, if requested, contains an anonymous internal name. If false, the initiator has been authenticated normally.
	GSS_C_PROT_READY_FLAG If true, the protection services, as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG, are available for use. If false, they are available only if the context is fully established, that is, if the <i>open</i> parameter is non-zero.
	<pre>GSS_C_TRANS_FLAG If true, resultant security context can be transferred to other processes through a call to gss_export_sec_context(). If false, the security context is not transferable.</pre>
locally_initiated	Non-zero if the invoking application is the context initiator. Specify NULL if the parameter is not required.
open	Non-zero if the context is fully established; zero if a context-establishment token is expected from the peer application. Specify NULL if the parameter is not required.

**RETURN VALUES** | gss inquire context() returns one of the following status codes:

GSS_S_COMPLETE	Successful completion.
GSS_S_NO_CONTEXT	The referenced context could not be accessed.
GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64–bit)
MT-Level	Safe

#### SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_context\_time(3GSS), gss\_delete\_sec\_context(3GSS), gss\_export\_sec\_context(3GSS), gss\_import\_sec\_context(3GSS), gss\_init\_sec\_context(3GSS), gss\_process\_context\_token(3GSS), gss\_wrap(3GSS), gss\_wrap\_size\_limit(3GSS), attributes(5)

gss\_inquire\_cred(3GSS)

NAME	gss_inquire_cred – obtain information about a credential		
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	<pre>OM_uint32 gss_inquire_cred(OM_uint32 *minor_status, const     gss_cred_id_t cred_handle, gss_name_t *name, OM_uint32 *lifetime,     gss_cred_usage_t *cred_usage, gss_OID_set *mechanisms);</pre>		
DESCRIPTION	Use the gss_inqu	uire_cred() function	n to obtain information about a credential.
PARAMETERS	The parameter des	scriptions for gss_acc	<pre>guire_cred() follow:</pre>
	minor_status	A mechanism specifi	c status code.
	cred_handle		to the target credential. Specify TIAL to inquire about the default initiator
	name	associated with this 1	ntity the credential asserts. Any storage name should be freed by the application after release_name(3GSS).
	lifetime	If the credential has e	ds for which the credential will remain valid. expired, this parameter will be set to zero. parameter is not required.
	cred_usage	contain one of the fol	nay be used. The <i>cred_usage</i> parameter may llowing values: GSS_C_INITIATE, GSS_C_BOTH. Specify NULL if this parameter
	mechanisms	the returned OID-set	ns which the credential supports. Storage for must be freed by the application after use by se_oid_set(3GSS). Specify NULL if this nired.
<b>RETURN VALUES</b>	gss_acquire_c	red() may return the	following status codes:
	GSS_S_COMPLETE	2	Successful completion.
	GSS_S_NO_CRED		The referenced credentials could not be accessed.
	GSS_S_DEFECTIV	/E_CREDENTIAL	The referenced credentials were invalid.
	GSS_S_CREDENT	IALS_EXPIRED	The referenced credentials have expired. If the <i>lifetime</i> parameter was not passed as NULL, it will be set to 0.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i>

gss\_inquire\_cred(3GSS)

parameter details the error condition.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

SEE ALSO

gss\_release\_name(3GSS), gss\_release\_oid\_set(3GSS), attributes(5)

## gss\_inquire\_cred\_by\_mech(3GSS)

NAME	gss_inquire_cred_by_mech - obtain per-mechanism information about a credential		
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	<pre>OM_uint32 gss_inquire_cred_by_mech(OM_uint32 *minor_status, const gss_cred_id_t cred_handle, const gss_OID mech_type, gss_name_t *name, OM_uint32 *initiator_lifetime, OM_uint32 *acceptor_lifetime, gss_cred_usage_t *cred_usage);</pre>		
DESCRIPTION	The gss_inquire credential.	e_cred_by_mech() c	btains per-mechanism information about a
PARAMETERS	The parameter des	scriptions for gss_inq	uire_cred_by_mech() follow:
	minor_status	A mechanism specific	e status code.
	cred_handle		o the target credential. Specify TAL to inquire about the default initiator
	mech_type	The mechanism for w	which the information should be returned.
	name	associated with this n	ntity the credential asserts. Any storage <i>ame</i> must be freed by the application after release_name(3GSS).
	initiator_lifetime	security contexts und can no longer be used for this mechanism is	ds that the credential is capable of initiating er the specified mechanism. If the credential l to initiate contexts, or if the credential usage GSS_C_ACCEPT, this parameter will be set this parameter is not required.
	acceptor_lifetime	security contexts und can no longer be used for this mechanism is	ds that the credential is capable of accepting er the specified mechanism. If the credential l to accept contexts, or if the credential usage GSS_C_INITIATE, this parameter will be h if this parameter is not required.
	cred_usage	The cred_usage param	hay be used with the specified mechanism. eter may contain one of the following values: SS_C_ACCEPT, or GSS_C_BOTH. Specify er is not required.
<b>RETURN VALUES</b>	gss_inquire_c	ced_by_mech() may	return the following status codes:
	GSS_S_COMPLETE	2	Successful completion.
	GSS_S_NO_CRED		The referenced credentials cannot be accessed.
	GSS_S_DEFECTIV	/E_CREDENTIAL	The referenced credentials are invalid

248 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

	gss_inquire_cred_by_mech(3GSS)
GSS_S_CREDENTIALS_EXPIRED	The credentials cannot be added because they have expired.
GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.

#### ATTRIBUTES S

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** gss\_release\_name(3GSS),, attributes(5)

## gss\_inquire\_mechs\_for\_name(3GSS)

NAME	gss_inquire_mechs_for_name - list mechanisms that support the specified name-type				
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>				
	<pre>OM_uint32 gss_inquire_mechs_for_name(OM_uint32 *minor_status, const gss_name_t input_name, gss_OID_set *mech_types);</pre>				
DESCRIPTION	The gss_inquire_mechs_for_name() function returns the set of mechanisms supported by the GSS-API that may be able to process the specified name. Each mechanism returned will recognize at least one element within the internal name.				
	Some implementations of the GSS-API may perform this test by checking nametype information contained within the passed name and registration information provided by individual mechanisms. This means that the <i>mech_types</i> set returned by the function may indicate that a particular mechanism will understand the name, when in fact the mechanism would refuse to accept the name as input to gss_canonicalize_name(3GSS), gss_init_sec_context(3GSS), gss_acquire_cred(3GSS), or gss_add_cred(3GSS), due to some property of the name itself rather than the name-type. Therefore, this function should be used only as a pre-filter for a call to a subsequent mechanism-specific function.				
PARAMETERS	The parameter descriptions for gss_inquire_mechs_for_name() follow in alphabetical order:				
	minor_status Mechanism		a-specific status code.		
	<i>input_name</i> The name to which the inquiry relates.				
	<i>mech_types</i> Set of mechanisms that may support the specified name. The returned OID set must be freed by the caller after use with a call to gss_release_oid_set(3GSS).				
<b>RETURN VALUES</b>	The gss_inquire_mechs_for_name() function may return the foll codes:		_for_name() function may return the following status		
	GSS_S_COMPLETE		Successful completion.		
	GSS_S_BAD_NAME		The <i>input_name</i> parameter was ill-formed.		
	GSS_S_BAD_NAMETYPE		The <i>input_name</i> parameter contained an invalid or unsupported type of name.		
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.		
ATTRIBUTES	See attribut	es(5) for des	scriptions of the following attributes:		

gss\_inquire\_mechs\_for\_name(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE	
Availability	SUNWgss (32-bit)	
	SUNWgssx (64-bit)	
MT-Level	Safe	

#### SEE ALSO gss\_acquire\_cred(3GSS), gss\_add\_cred(3GSS), gss\_canonicalize\_name(3GSS), gss\_init\_sec\_context(3GSS), gss\_release\_oid\_set(3GSS), attributes(5)

## gss\_inquire\_names\_for\_mech(3GSS)

NAME	gss_inquire_names_for_mech – list the name-types supported by the specified mechanism				
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h=""></gssapi></pre>				
	<pre>OM_uint32 gss_inquire_names_for_mech(OM_uint32 *minor_status, const gss_OID mechanism, gss_OID_set *name_types);</pre>				
DESCRIPTION	The gss_inquire_names_for_mech() function returns the set of name-types supported by the specified mechanism.				
PARAMETERS	The parameter descriptions for gss_inquire_names_for_mech() follow:				
	<i>minor_status</i> A mechanism-specific status code.				
	mechanism	The mechanism to be inter	rrogated.		
	name_types		ed by the specified mechanism. The freed by the application after use with a call et (3GSS).		
<b>RETURN VALUES</b>	The gss_inquire_names_for_mech() function may return the following values:				
	GSS_S_COMPLETE Successful completion.				
	GSS_S_FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
		ATTRIBUTE TYPE	ATTRIBUTE VALUE		
	Availability		SUNWgss (32-bit)		
			SUNWgssx (64-bit)		
	MT-Level		Safe		
SEE ALSO	gss_release_oid_set(3GSS), attributes(5) GSS-API Programming Guide				

252 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

NAME	gss_oid_to_str - co	onvert an OID to a strin	g
SYNOPSIS	0	lgss [ library .	0
	gss_oid_to_sta gss_buffe	—	<pre>status, const gss_OID *oid,</pre>
DESCRIPTION	The gss_oid_to_str() function converts a GSS-API OID structure to a string. You can use the function to convert the name of a mechanism from an OID to a simple string. This function is a convenience function, as is its complementary function, gss_str_to_oid(3GSS).		
	If an OID must be created, use gss_create_empty_oid_set(3GSS) and gss_add_oid_set_member()(3GSS) to create it. OIDs created in this way must be released with gss_release_oid_set(3GSS). However, it is strongly suggested that applications use the default GSS-API mechanism instead of creating an OID for a specific mechanism.		
PARAMETERS	The parameter des	scriptions for gss_oid	_to_str() are as follows:
	minor_status	Status code returned l	by underlying mechanism.
	oid	GSS-API OID structur	re to convert.
	oid_str	String to receive conv	erted OID.
<b>RETURN VALUES</b>	gss_oid_to_st	c() returns one of the f	ollowing status codes:
	GSS_S_CALL_INA	ACCESSIBLE_READ	A required input parameter could not be read.
	GSS_S_CALL_INA	ACCESSIBLE_WRITE	A required output parameter could not be written.
	GSS_S_COMPLET	Ξ	Successful completion.
			The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe

#### gss\_oid\_to\_str(3GSS)

SEE ALSO	gss_add_oid_set_member()(3GSS),gss_create_empty_oid_set(3GSS),
	<pre>gss_release_oid_set(3GSS), gss_str_to_oid(3GSS), attributes(5)</pre>

GSS-API Programming Guide

**WARNINGS** This function is included for compatibility only with programs using earlier versions of the GSS-API and should not be used for new programs. Other implementations of the GSS-API might not support this function, so portable programs should not rely on it. Sun might not continue to support this function.

#### gss process context token(3GSS) gss\_process\_context\_token - pass asynchronous token to security service NAME | **SYNOPSIS** cc -flag ... file ... -lgss [ library ... ] #include <gssapi/gssapi.h> OM uint32 gss process context token (OM uint32 \*minor\_status, const gss ctx id t context\_handle, const gss buffer t token\_buffer); DESCRIPTION The gss process context token() function provides a way to pass an asynchronous token to the security service. Most context-level tokens are emitted and processed synchronously by gss init sec context() and gss accept sec context(), and the application is informed as to whether further tokens are expected by the GSS C CONTINUE NEEDED major status bit. Occasionally, a mechanism might need to emit a context-level token at a point when the peer entity is not expecting a token. For example, the initiator's final call to gss init sec context() may emit a token and return a status of GSS S COMPLETE, but the acceptor's call to gss accept sec context() might fail. The acceptor's mechanism might want to send a token containing an error indication to the initiator, but the initiator is not expecting a token at this point, believing that the context is fully established. gss process context token() provides a way to pass such a token to the mechanism at any time. This function is provided for compatibility with the GSS-API version 1. Because gss delete sec context() no longer returns a valid output\_token to be sent to gss process context token(), applications using a newer version of the GSS-API do not need to rely on this function. PARAMETERS The parameter descriptions for qss process context token() are as follows: minor\_status A mechanism-specific status code. context\_handle Context handle of context on which token is to be processed. token\_buffer Token to process. **RETURN VALUES** gss process context token() returns one of the following status codes: GSS S COMPLETE Successful completion. Indicates that consistency checks performed GSS S DEFECTIVE TOKEN on the token failed. GSS S NO CONTEXT The context\_handle did not refer to a valid context. GSS S FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the *minor\_status* parameter details the error condition. ATTRIBUTES See attributes(5) for descriptions of the following attributes:

#### gss\_process\_context\_token(3GSS)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT Level	Safe

SEE ALSO gss\_accept\_sec\_context(3GSS), gss\_delete\_sec\_context(3GSS), gss\_init\_sec\_context(3GSS), attributes(5)

GSS-API Programming Guide

			8
NAME	gss_release_buffer	– free buffer storage allo	cated by a GSS-API function
SYNOPSIS		lgss [ library	. 1
	<pre>#include <gssapi <="" pre=""></gssapi></pre>		
	OM_uint32 <b>gss</b> _s gss_buffer	<b>release_buffer</b> (OM_` t <i>buffer</i> );	uint32 * <i>minor_status</i> ,
DESCRIPTION	The gss_release_buffer() function frees buffer storage allocated by a GSS-API function. The gss_release_buffer() function also zeros the length field in the descriptor to which the buffer parameter refers, while the GSS-API function sets the pointer field in the descriptor to NULL. Any buffer object returned by a GSS-API function may be passed to gss_release_buffer(), even if no storage is associated with the buffer.		
PARAMETERS	The parameter des	criptions for gss_relea	ase_buffer() follow:
	minor_status	Mechanism-specific stat	tus code.
	buffer		with the buffer will be deleted. The object will not be freed; however, its length
<b>RETURN VALUES</b>	The gss_release	_buffer() function ma	ay return the following status codes:
	GSS_S_COMPLETE	Successful completion	on
	GSS_S_FAILURE	specific GSS status c	hanism detected an error for which no ode is defined. The mechanism-specific by means of the <i>minor_status</i> parameter dition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTR	IBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe
SEE ALSO	attributes(5) GSS-API Programm	ning Guide	

### gss\_release\_cred(3GSS)

NAME	gss_release_cred -	discard a cr	redential handle
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	OM_uint32 <b>gss</b> _ *cred_handle		cred(OM_uint32 * <i>minor_status</i> , gss_cred_id_t
DESCRIPTION	The gss_release_cred() function informs the GSS-API that the specified credential handle is no longer required by the application and frees the associated resources. The <i>cred_handle</i> parameter is set to GSS_C_NO_CREDENTIAL when this call completes successfully.		
PARAMETERS	The parameter des	scriptions fo	rgss_release_cred() follow:
	minor_status	A mechanism specific status code.	
	cred_handle	An opaque handle that identifies the credential to be released. If GSS_C_NO_CREDENTIAL is specified, the gss_release_cred() function will complete successfully, but it will do nothing.	
<b>RETURN VALUES</b>	gss_release_cred() may return the following status codes:		return the following status codes:
	GSS_S_COMPLETE	6	Successful completion.
	GSS_S_NO_CRED		The referenced credentials cannot be accessed.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(	5) for descri	ptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** attributes(5)

GSS-API Programming Guide

258 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

NAME	gss_release_name – discard an internal-form name		
SYNOPSIS	<pre>cc [flag] filelgss [library] #include <gssapi gssapi.h<="" pre=""></gssapi></pre>		
	<pre>OM_uint32 gss_release_name(OM_uint32 *minor_status, gss_name_t *name);</pre>		
DESCRIPTION	The gss_release_name() function frees GSS-API-allocated storage associated with an internal-form name. The <i>name</i> is set to GSS_C_NO_NAME on successful completion of this call.		
PARAMETERS	The parameter descriptions for gss_release_name() follow:		
	<i>minor_status</i> A mechanism-specific status code.		
	<i>name</i> The name to be deleted.		
<b>RETURN VALUES</b>	The gss_release_name() function may return the following status codes:		
	GSS_S_COMPLETE Successful completion.		
	GSS_S_BAD_NAME The <i>name</i> parameter did not contain a valid name.		
	GSS_S_FAILURE The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64–bit)
MT-Level	Safe

**SEE ALSO** attributes(5)

GSS-API Programming Guide

gss\_release\_oid(3GSS)

0 - 1	•		
NAME	gss_release_oid - re	elease an object identifie	r
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	<pre>gss_release_oid(OM_uint32 *minor_status, const gss_OID *oid);</pre>		
DESCRIPTION	The gss_release_oid() function deletes an OID. Such an OID might have been created with gss_str_to_oid().		
	Since creating and deleting individual OIDs is discouraged, it is preferable to use gss_release_oid_set() if it is necessary to deallocate a set of OIDs.		
PARAMETERS	The parameter des	criptions for gss_relea	ase_oid() are as follows:
	minor_status	A mechanism-specific s	tatus code.
	oid	The object identifier of	the mechanism to be deleted.
<b>RETURN VALUES</b>	gss_release_oi	d() returns one of the f	ollowing status codes:
	GSS_S_COMPLETE	Successful completi	on.
	GSS_S_FAILURE	specific GSS status of	chanism detected an error for which no code is defined. The mechanism-specific d by means of the <i>minor_status</i> parameter adition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTR	IBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT Level		Safe
			Sale
SEE ALSO			
SEE ALSO	gss_release_oi GSS-API Programm		
SEE ALSO WARNINGS	GSS-API Programm This function is inc of the GSS-API and the GSS-API might	ning Guide luded for compatibility l should not be used for	

NAME	gss_release_oid_set – object	free storage associated	d with a GSS-API-generated gss_OID_set
SYNOPSIS	<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi></pre>		
	OM_uint32 gss_re *set);	lease_oid_set(OM	_uint32 * <i>minor_status</i> , gss_OID_set
DESCRIPTION	The gss_release_oid_set() function frees storage associated with a GSS-API-generated gss_OID_set object. The <i>set</i> parameter must refer to an OID-set that was returned from a GSS-API function. The gss_release_oid_set() function will free the storage associated with each individual member OID, the OID <i>set</i> 's elements array, and gss_OID_set_desc.		
	gss_OID_set is set f function.	to GSS_C_NO_OID_SE	ET on successful completion of this
PARAMETERS	The parameter descri	ptions for gss_relea	<pre>use_oid_set() follow:</pre>
	minor_status A	mechanism-specific s	tatus code
	set St	orage associated with	the gss_OID_set will be deleted
<b>RETURN VALUES</b>	Thegss_release_c	oid_set() function m	nay return the following status codes:
	GSS_S_COMPLETE	Successful completion	on
	GSS_S_FAILURE	specific GSS status c	hanism detected an error for which no ode is defined. The mechanism-specific by means of the <i>minor_status</i> parameter dition.
ATTRIBUTES	See attributes(5) f	or descriptions of the	following attributes:
	ATTRIBL	ITE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64–bit)
	MT-Level		Safe
SEE ALSO	attributes(5)		

GSS-API Programming Guide

Networking Library Functions 261

#### gss\_str\_to\_oid(3GSS)

NAME	gss_str_to_oid - co	onvert a string to an OI	D
SYNOPSIS	cc -flag <i>file</i> #include <gssapi< th=""><th>2lgss [ library . /gssapi.h&gt;</th><th>]</th></gssapi<>	2lgss [ library . /gssapi.h>	]
	<pre>OM_uint32 gss_str_to_oid(OM_uint32 *minor_status, const gss_buffer_t</pre>		
DESCRIPTION	The gss_str_to_oid() function converts a string to a GSS-API OID structure. You can use the function to convert a simple string to an OID to . This function is a convenience function, as is its complementary function, gss_oid_to_str(3GSS).		
	OIDs created with gss_str_to_oid() must be deallocated through gss_release_oid(3GSS), if available. If an OID must be created, use gss_create_empty_oid_set(3GSS) and gss_add_oid_set_member()(3GSS) to create it. OIDs created in this way must be released with gss_release_oid_set(3GSS). However, it is strongly suggested that applications use the default GSS-API mechanism instead of creating an OID for a specific mechanism.		
PARAMETERS	The parameter des	scriptions for gss_str	_to_oid() are as follows:
	minor_status	Status code returned	by underlying mechanism.
	oid	GSS-API OID structu	re to receive converted string.
	oid_str	String to convert.	
<b>RETURN VALUES</b>	gss_str_to_oid	d() returns one of the	following status codes:
	GSS_S_CALL_INA	ACCESSIBLE_READ	A required input parameter could not be read.
	GSS_S_CALL_INA	ACCESSIBLE_WRITE	A required output parameter could not be written.
	GSS_S_COMPLET	Ξ	Successful completion.
	GSS_S_FAILURE		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64–bit)

<sup>262</sup> man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

gss\_str\_to\_oid(3GSS)

		800_011_00_011(0.000)			
	ATTRIBUTE TYPE	ATTRIBUTE VALUE			
	MT Level	Safe			
SEE ALSO	gss_add_oid_set_member()(3GSS),gs gss_oid_to_str(3GSS),gss_release_				
	GSS-API Programming Guide				
WARNINGS	This function is included for compatibility only with programs using earlier versions of the GSS-API and should not be used for new programs. Other implementations of the GSS-API might not support this function, so portable programs should not rely on it. Sun might not continue to support this function.				

gss\_test\_oid\_set\_member(3GSS)

NAME	gss_test_oid_set	t_member – interrogate an	object identifier set
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
			er(OM_uint32 * <i>minor_status</i> , const 0_set <i>set</i> , int * <i>present</i> );
DESCRIPTION	The gss_test_oid_set_member() function interrogates an object identifier set to determine if a specified object identifier is a member. This function should be used with OID sets returned by gss_indicate_mechs(3GSS), gss_acquire_cred(3GSS), and gss_inquire_cred(3GSS), but it will also work with user-generated sets.		
PARAMETERS	The parameter of	descriptions for gss_test	_oid_set_member() follow:
	minor_status	A mechanism-specific stat	us code
	member	An object identifier whose	presence is to be tested
	set	An object identifier set.	
	present	The value of <i>present</i> is non the set; if not, the value of	-zero if the specified OID is a member of <i>present</i> is zero.
<b>RETURN VALUES</b>	The gss_test	_oid_set_member() fund	ction may return the following status codes:
	GSS_S_COMPLE	TE Successful completi	on
	GSS_S_FAILUF	specific GSS status	chanism detected an error for which no code is defined. The mechanism-specific d by means of the <i>minor_status</i> parameter ndition.
ATTRIBUTES	See attribute	es(5) for descriptions of the	following attributes:
	A	TTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWgss (32-bit)
			SUNWgssx (64-bit)
	MT-Level		Safe
SEE ALSO		_cred(3GSS), gss_indica _cred(3GSS), attributes mmning Guide	

264 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

NAME	gss_wrap – verify a mes	sage with attache	ed cryptographic message
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	context_handle, co	onst gss_buff	<pre>*minor_status, const gss_ctx_id_t er_t input_message_buffer, gss_buffer_t state, gss_qop_t *qop_state);</pre>
DESCRIPTION	gss_wrap(3GSS) back t parameter indicates who	to a usable form, to a usable form, the message	message previously protected by verifying the embedded MIC. The <i>conf_state</i> e was encrypted; the <i>qop_state</i> parameter vas used to provide the confidentiality and
		ovide secure fram	ay wish to use tokens emitted by ing, the GSS-API supports the wrapping and
PARAMETERS	The parameter descripti	ons for gss_unw	rap() follow:
	minor_status	The status code	returned by the underlying mechanism.
	context_handle	Identifies the co	ntext on which the message arrived.
	input_message_buffer	The message to	be protected.
	output_message_buffer	associated with	ceive the unwrapped message. Storage this buffer must be freed by the application call to gss_release_buffer(3GSS).
	conf_state	integrity protect	<i>onf_state</i> is non-zero, then confidentiality and tion were used. If the value is zero, only was used. Specify NULL if this parameter is
	qop_state	Specifies the qu this parameter i	ality of protection provided. Specify NULL if s not required.
<b>RETURN VALUES</b>	gss_unwrap() may re	turn the followin	g status codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_DEFECTIVE_TO	OKEN	The token failed consistency checks.
	GSS_S_BAD_SIG		The MIC was incorrect.
	GSS_S_DUPLICATE_TC	OKEN	The token was valid, and contained a correct MIC for the message, but it had already been processed.
	GSS_S_OLD_TOKEN		The token was valid, and contained a correct MIC for the message, but it is too old to check for duplication.

gss\_unwrap(3GSS)

GSS_S_UNSEQ_TOKEN	The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; a later token has already been received.
GSS_S_GAP_TOKEN	The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; an earlier expected token has not yet been received.
GSS_S_CONTEXT_EXPIRED	The context has already expired.
GSS_S_NO_CONTEXT	The <i>context_handle</i> parameter did not identify a valid context.
GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

**SEE ALSO** gss\_release\_buffer(3GSS), gss\_wrap(3GSS), attributes(5)

GSS-API Programming Guide

NAME	gss_verify_mic -	verify integrity of a rece	eived message
SYNOPSIS	<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi></pre>		
	context_han		<pre>ht32 *minor_status, const gss_ctx_id_t er_t message_buffer, const _qop_t *qop_state);</pre>
DESCRIPTION	token parameter,	, fits the supplied messag	es that a cryptographic MIC, contained in the ge. The <i>qop_state</i> parameter allows a message stection that was applied to the message.
	gss_wrap(3GSS		ay wish to use tokens emitted by ing, the GSS-API supports the calculation and ssages.
PARAMETERS	The parameter d	escriptions for gss_ver	<pre>ify_mic() follow:</pre>
	minor_status	The status code returne	ed by the underlying mechanism.
	context_handle	Identifies the context o	n which the message arrived.
	message_buffer	The message to be veri	fied.
	token_buffer	The token associated w	rith the message.
	qop_state	Specifies the quality of NULL if this parameter	protection gained from the MIC. Specify is not required.
<b>RETURN VALUES</b>	gss_verify_m	ic() may return the foll	owing status codes:
	GSS_S_COMPLE	TE	Successful completion.
	GSS_S_DEFECT	IVE_TOKEN	The token failed consistency checks.
	GSS_S_BAD_SI	G	The MIC was incorrect.
	GSS_S_DUPLIC	ATE_TOKEN	The token was valid and contained a correct MIC for the message, but it had already been processed.
	GSS_S_OLD_TO	KEN	The token was valid and contained a correct MIC for the message, but it is too old to check for duplication.
	GSS_S_UNSEQ_	TOKEN	The token was valid and contained a correct MIC for the message, but it has been verified out of sequence; a later token has already been received.
	GSS_S_GAP_TO	KEN	The token was valid and contained a correct MIC for the message, but it has been verified out of sequence; an earlier expected token has not yet been received.
			Networking Library Eurocions 267

Networking Library Functions 267

### gss\_verify\_mic(3GSS)

	GSS_S_CONTEXT_EXPIRED	The context has already expired.
	GSS_S_NO_CONTEXT	The <i>context_handle</i> parameter did not identify a valid context.
		The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5) for descriptions of the	e following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE

Availability	SUNWgss (32-bit)
	SUNWgssx (64-bit)
MT-Level	Safe

### **SEE ALSO** gss\_wrap(3GSS), attributes(5)

GSS-API Programming Guide

268 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

			$O = I \langle \rangle$
NAME	gss_wrap – attach a cry	ptographic messa	ge
SYNOPSIS	<pre>cc -flag filelgss [ library ] #include <gssapi gssapi.h=""></gssapi></pre>		
	context_handle, i	nt conf_req_flag , input_message_bu <u>f</u>	ninor_status, const gss_ctx_id_t gss_qop_t qop_req, const fer, int *conf_state, gss_buffer_t
DESCRIPTION	specified input_message.	. The <i>output_messi</i> allows a choice be	ptographic MIC and optionally encrypts the age contains both the MIC and the message. etween several cryptographic algorithms, if
			ay wish to use tokens emitted by the GSS-API supports the wrapping of
PARAMETERS	The parameter descript	ions for gss_wra	up() follow:
	minor_status	The status code	returned by the underlying mechanism.
	context_handle	Identifies the co	ontext on which the message will be sent.
	conf_req_flag	and integrity se	<i>onf_req_flag</i> is non-zero, both confidentiality prvices are requested. If the value is zero, then ervice is requested.
	qop_req	mechanism-spe <i>qop_req</i> to GSS_ protection strer	quired quality of protection. A cific default may be requested by setting C_QOP_DEFAULT. If an unsupported gth is requested, gss_wrap() will return a GSS_S_BAD_QOP.
	input_message_buffer	The message to	be protected.
	conf_state	origin authention applied. If the v	<i>onf_state</i> is non-zero, confidentiality, data cation, and integrity services have been value is zero, then integrity services have been vILL if this parameter is not required.
	output_message_buffer	associated with application afte	ceive the protected message. Storage this message must be freed by the r use with a call to _buffer(3GSS).
<b>RETURN VALUES</b>	gss_wrap() may retu	rn the following s	tatus codes:
	GSS_S_COMPLETE		Successful completion.
	GSS_S_CONTEXT_EXP	IRED	The context has already expired.

# gss\_wrap(3GSS)

	GSS_S_NO_CONTEXT	The <i>context_handle</i> parameter did not identify a valid context.
	GSS_S_BAD_QOP	The specified QOP is not supported by the mechanism.
	GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.
ATTRIBUTES	See attributes(5) for descriptions of t	he following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWgss (32-bit)
		SUNWgssx (64-bit)
	MT-Level	Safe
SEE ALSO	gss_release_buffer(3GSS), attrib GSS-API Programming Guide	utes(5)

NAME		nit – allow application t bken of a specified max	to determine maximum message size with imum size
SYNOPSIS	cc -flag <i>file</i> lgss [ library ] #include <gssapi gssapi.h=""></gssapi>		
	gss_ctx_i	d_t context_handle, in	token(OM_uint32 *minor_status, const nt conf_req_flag, gss_qop_t qop_req, hint32 *max_input_size);
DESCRIPTION	The gss_wrap_size_limit() function allows an application to determine the maximum message size that, if presented to gss_wrap() with the same <i>conf_req_flag</i> and <i>qop_req</i> parameters, results in an output token containing no more than <i>req_output_size</i> bytes. This call is intended for use by applications that communicate over protocols that impose a maximum message size. It enables the application to fragment messages prior to applying protection. The GSS-API detects invalid QOP values when gss_wrap_size_limit() is called. This routine guarantees only a maximum message size, not the availability of specific QOP values for message protection.		
	Successful completion of gss_wrap_size_limit() does not guarantee that gss_wrap() will be able to protect a message of length <i>max_input_size</i> bytes, since this ability might depend on the availability of system resources at the time that gss_wrap() is called.		
PARAMETERS	The parameter des	scriptions for gss_wra	p_size_limit() are as follows:
	minor_status	A mechanism-specific	c status code.
	context_handle	A handle that refers t be sent.	o the security over which the messages will
	conf_req_flag		s_wrap() will be asked to apply confidential to integrity protection. See gss_wrap(3GSS)
	qop_req		protection that gss_wrap() will be asked to ap(3GSS) for more details.
	req_output_size	The desired maximur	n size for tokens emitted by gss_wrap().
	<i>max_input_size</i> The maximum input message size that can be presented to gss_wrap() to guarantee that the emitted token will be no larger than <i>req_output_size</i> bytes.		
<b>RETURN VALUES</b>	gss_wrap_size_	_limit() returns one	of the following status codes:
	GSS_S_COMPLETE	2	Successful completion.
	GSS_S_NO_CONTE	EXT	The referenced context could not be accessed.
	GSS_S_CONTEXT_	EXPIRED	The context has expired.

Networking Library Functions 271

gss\_wrap\_size\_limit(3GSS)

GSS_S_BAD_QOP	The specified QOP is not supported by the mechanism.
GSS_S_FAILURE	The underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code reported by means of the <i>minor_status</i> parameter details the error condition.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWgss (32-bit)
	SUNWgssx (64–bit)
MT Level	Safe

**SEE ALSO** gss\_wrap(3GSS), attributes(5)

GSS-API Programming Guide

272 man pages section 3: Networking Library Functions • Last Revised 24 Apr 2000

NAME	htonl, htons, ntohl, ntohs – convert values between host and network byte order		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <arpa inet.h=""></arpa></pre>		
	<pre>uint32_t htonl (uint32_t hostlong);</pre>		
	<pre>uint16_t htons(uint16_t hostshort);</pre>		
	<pre>uint32_t ntohl(uint32_t netlong);</pre>		
	<pre>uint16_t ntohs(uint16_t netshort);</pre>		
DESCRIPTION	These functions convert 16-bit and 32-bit quantities between network byte order and host byte order.		
	The uint32_t and uint16_t types are m <inttypes.h>.</inttypes.h>	ade available by inclusion of	
USAGE	These functions are most often used in conjunction with Internet addresses and ports as returned by gethostent(3XNET) and getservent(3XNET).		
	On some architectures these functions are defined as macros that expand to the value of their argument.		
RETURN VALUES	The htonl() and htons() functions return the argument value converted from host to network byte order.		
	The ntohl() and ntohs() functions return the argument value converted from network to host byte order.		
ERRORS	No errors are defined.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level MT-Safe		
SEE ALSO	<pre>endhostent(3XNET), endservent(3XNET), attributes(5)</pre>		

#### if\_nametoindex(3NSL)

NAME	if_nametoindex, if_indextoname, if_nameindex, if_freenameindex – routines to map Internet Protocol network interface names and interface indexes		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <net if.h=""></net></pre>		
	unsigned int <b>if_nametoindex</b> (const char * <i>ifname</i> );		
	<pre>char *if_indextoname(unsigned int ifindex, char *ifname);</pre>		
	<pre>struct if_nameindex *if_nameindexvoid););</pre>		
	<pre>void if_freenameindex(struct if_nameindex *ptr);</pre>		
DESCRIPTION	This API defines two functions that map between an Internet Protocol network interface name and index, a third function that returns all the interface names and indexes, and a fourth function to return the dynamic memory allocated by the previous function.		
	Network interfaces are normally known by names such as "le0", "sl1", "ppp2", and the like. The <i>ifname</i> argument must point to a buffer of at least IF_NAMESIZE bytes into which the interface name corresponding to the specified index is returned. IF_NAMESIZE is defined in <net if.h=""> and its value includes a terminating null byte at the end of the interface name.</net>		
	<pre>if_nametoindex() The if_nametoindex() function returns the interface index corresponding to the interface name pointed to by the <i>ifname</i> pointer. If the specified interface name does not exist, the return value is 0, and errno is set to ENXIO. If there was a system error, such as running out of memory, the return value is 0 and errno is set to the proper value, for example, ENOMEM.</pre>		
	<pre>if_indextoname() The if_indextoname() function maps an interface index into its corresponding name. This pointer is also the return value of the function. If there is no interface corresponding to the specified index, NULL is returned, and errno is set to ENXIO, if there was a system error, such as running out of memory, if_indextoname() returns NULL and errno would be set to the proper value, for example, ENOMEM.</pre>		
	<pre>*if_nameindex() The if_nameindex() function returns an array of if_nameindex structures, one structure per interface. The if_nameindex structure holds the information about a single interface and is defined when the <net if.h=""> header is included:</net></pre>		
	<pre>struct if_nameindex unsigned int if_index; /* 1, 2, */ char *if_name; /* null terminated name: "le0", */ };</pre>		
	The end of the array of structures is indicated by a structure with an if_index of 0 and an if_name of NULL. The function returns a null pointer upon an error and sets errno to the appropriate value. The memory used for this array of structures		

if\_nametoindex(3NSL)

	along with the interface names pointed to by the if_name members is obtained dynamically. This memory is freed by the if_freenameindex() function.		
	<pre>if_freenameindex() The if_freenameindex() function frees the dynamic memory that was allocated by if_nameindex(). The argument to this function must be a pointer that was returned by if_nameindex().</pre>		
PARAMETERS	ifname	interface nar	ne.
	ifindex	interface ind	ex.
	ptr	pointer retur	<pre>rned by if_nameindex().</pre>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE		ATTRIBUTE VALUE

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
MT Level	MT Safe
Interface Stability	Standard

**SEE ALSO** if config(1M), attributes(5), if(7P)

Networking Library Functions 275

#### if\_nametoindex(3XNET)

nametoindex(3XINET)		
NAME	if_nametoindex, if_indextoname, if_nameindex, if_freenameindex – functions to map Internet Protocol network interface names and interface indexes	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <net if.h=""></net></pre>	
	unsigned int <b>if_nametoindex</b> (const char * <i>ifname</i> );	
	<pre>char *if_indextoname(unsigned int ifindex, char *ifname);</pre>	
	<pre>struct if_nameindex *if_nameindexvoid););</pre>	
	<pre>void if_freenameindex(struct if_nameindex *ptr);</pre>	
DESCRIPTION	This API defines two functions that map between an Internet Protocol network interface name and index, a third function that returns all the interface names and indexes, and a fourth function to return the dynamic memory allocated by the previous function.	
	Network interfaces are normally known by names such as "le0", "sl1", "ppp2", and the like. The <i>ifname</i> argument must point to a buffer of at least IF_NAMESIZE bytes into which the interface name corresponding to the specified index is returned. IF_NAMESIZE is defined in <net if.h=""> and its value includes a terminating null byte at the end of the interface name.</net>	
	<pre>if_nametoindex() The if_nametoindex() function returns the interface index corresponding to the interface name pointed to by the <i>ifname</i> pointer. If the specified interface name does not exist, the return value is 0, and errno is set to ENXIO. If there was a system error, such as running out of memory, the return value is 0 and errno is set to the proper value, for example, ENOMEM.</pre>	
	<pre>if_indextoname() The if_indextoname() function maps an interface index into its corresponding name. This pointer is also the return value of the function. If there is no interface corresponding to the specified index, NULL is returned, and errno is set to ENXIO, if there was a system error, such as running out of memory, if_indextoname() returns NULL and errno would be set to the proper value, for example, ENOMEM.</pre>	
	<pre>*if_nameindex() The if_nameindex() function returns an array of if_nameindex structures, one structure per interface. The if_nameindex structure holds the information about a single interface and is defined when the <net if.h=""> header is included:</net></pre>	
	<pre>struct if_nameindex {     unsigned int if_index; /* 1, 2, */     char *if_name; /* null terminated name: "le0", */ };</pre>	
	The end of the array of structures is indicated by a structure with an if_index of 0 and an if_name of NULL. The function returns a null pointer upon an error and sets errno to the appropriate value. The memory used for this array of structures	

if\_nametoindex(3XNET)

 along with the interface names pointed to by the if\_name members is obtained dynamically. This memory is freed by the if\_freenameindex() function.

 if\_freenameindex()

 The if\_freenameindex() function frees the dynamic memory that was allocated by if\_nameindex(). The argument to this function must be a pointer that was returned by if\_nameindex().

 PARAMETERS
 ifname interface name.

 ifindex
 interface index.

 ptr
 pointer returned by if\_nameindex().

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
MT Level	MT Safe
Interface Stability	Standard

**SEE ALSO** if config(1M), attributes(5), if (7P)

Networking Library Functions 277

#### inet(3SOCKET)

inet, inet6, inet_ntop, inet_pton, inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation		
NOPSIS cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""> #include <netinet in.h=""> #include <arpa inet.h=""></arpa></netinet></sys></sys>		
<pre>const char *inet_ntop(int af, const void *addr, char *cp, size_t size);</pre>		
<pre>int inet_pton(int af, const char *cp, void *addr);</pre>		
<pre>in_addr_t inet_addr(const char *cp);</pre>		
<pre>in_addr_t inet_network(const char *cp);</pre>		
<pre>struct in_addr inet_makeaddr(const int net, const int lna);</pre>		
<pre>int inet_lnaof(const struct in_addr in);</pre>		
<pre>int inet_netof(const struct in_addr in);</pre>		
<pre>char *inet_ntoa(const struct in_addr in);</pre>		
The inet_ntop() and inet_pton() routines can manipulate both IPv4 and IPv6 addresses, whereas inet_addr(), inet_network(), inet_makeaddr(), inet_lnaof(), inet_netof(), and inet_ntoa() can only manipulate IPv4 addresses.		
The inet_ntop() routine converts a numeric address into a string suitable for presentation. The <i>af</i> argument specifies the family of the address. This can be AF_INET or AF_INET6. The <i>addr</i> argument points to a buffer holding an IPv4 address if the <i>af</i> argument is AF_INET, or an IPv6 address if the <i>af</i> argument is AF_INET6; the address must be in network byte order. The <i>cp</i> argument points to a buffer where the routine will store the resulting string. The <i>size</i> argument. For IPv6 addresses, the buffer must be at least 46-octets. For IPv4 addresses, the buffer must be at least 46-octets. For IPv4 addresses, the buffer so the proper size to store IPv4 and IPv6 addresses in string form, the following two constants are defined in <netinet in.h="">:</netinet>		
<pre>#define INET_ADDRSTRLEN 16 #define INET_ADDRSTRLEN 46The inet_pton() routine converts an address in its standard text presentation form into its numeric binary form. The <i>af</i> argument specifies the family of the address. Currently the AF_INET and AF_INET6 address families are supported. The <i>cp</i> argument points to the string being passed in. The <i>addr</i> argument points to a buffer into which the routine stores the numeric address. The calling application must ensure that the buffer referred to by <i>addr</i> is large enough to hold the numeric address, at least 4 bytes for AF_INET or 16 bytes for AF_INET6.</pre>		

	The inet_addr() and inet_network() routines interpret character strings representing numbers expressed in the IPv4 standard '.' notation, returning numbers suitable for use as IPv4 addresses and IPv4 network numbers, respectively. The routine inet_makeaddr() takes an IPv4 network number and a local network address and constructs an IPv4 address from it. The routines inet_netof() and inet_lnaof() break apart IPv4 host addresses, returning the network number and local network address part, respectively. The inet_ntoa() routine returns a pointer to a string in the base 256 notation d.d.d.d. See INTERNET ADDRESSES. Internet addresses are returned in network order, bytes ordered from left to right. Network numbers and local address parts are returned as machine format integer values.
IPv6 Addresses	There are three conventional forms for representing IPv6 addresses as strings:
	1. The preferred form is x:x:x:x:x:x:x, where the 'x's are the hexadecimal values of the eight 16-bit pieces of the address, for example,
	1080:0:0:0:8:800:200C:417A Note that it is not necessary to write the leading zeros in an individual field. However, there must be at least one numeral in every field, except as described below.
	2. Due to some methods of allocating certain styles of IPv6 addresses, it will be common for addresses to contain long strings of zero bits. In order to make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. The use of "::" indicates multiple groups of 16-bits of zeros. The "::" can only appear once in an address. The "::" can also be used to compress the leading and/or trailing zeros in an address. For example,
	1080::8:800:200C:417A
	3. An alternative form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is x:x:x:x:x:d.d.d.d, where the 'x's are the hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the decimal values of the four low-order 8-bit pieces of the standard IPv4 representation address, for example,
	::FFFF:129.144.52.38 ::129.144.52.38 where "::FFFF:d.d.d.d" and "::d.d.d.d" are, respectively, the general forms of an IPv4–mapped IPv6 address and an IPv4–compatible IPv6 address. Note that the IPv4 portion must be in the "d.d.d.d" form. The following forms are invalid:
	::FFFF:d.d. ::FFFF:d.d ::d.d.d ::d.d The following form:

inet(3SOCKET)		
	: : FFFF:d is valid, however it is an unconventional representation of the IPv4–compatible IPv6 address,	
	::255.255.0.d while "::d" corresponds to the general IPv6 address " $0:0:0:0:0:0:0:d$ ".	
IPv4 Addresses	Values specified using '. ' notation take one of the following forms:	
	d.d.d d.d.d d.d dWhen four parts are specified, each is interpreted as a byte of data and assigned,	
	from left to right, to the four bytes of an IPv4 address.	
	When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as 128.net.host.	
	When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as net.host.	
	When only one part is given, the value is stored directly in the network address without any byte rearrangement.	
	With the exception of inet_pton(), numbers supplied as <i>parts</i> in '.' notation may be decimal, octal, or hexadecimal, as specified in the C language. For example, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal.	
	For IPv4 addresses, inet_pton() only accepts a string in the standard IPv4 dotted-decimal form:	
	d.d.d.dwhere each number has one to three digits with a decimal value between 0 and 255.	
RETURN VALUES	The inet_ntop() routine returns a pointer to the buffer containing a string if the conversion succeeds, and NULL otherwise. Upon failure, errno is set to EAFNOSUPPORT if the <i>af</i> argument is invalid or ENOSPC if the size of the result buffer is inadequate.	
	<pre>inet_pton() returns 1 if the conversion succeeds, 0 if the input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string, or -1 with errno set to EAFNOSUPPORT if the af argument is unknown.</pre>	
	The value -1 is returned by inet_addr() and inet_network() for malformed requests.	
<b>000</b>	0. Networking Library Eventions - Level Device d 0 New 4000	

280 man pages section 3: Networking Library Functions • Last Revised 3 Nov 1999

The routines inet\_netof() and inet\_lnaof() break apart IPv4 host addresses, returning the network number and local network address part, respectively.

The routine inet\_ntoa() returns a pointer to a string in the base 256 notation d.d.d.d described in INTERNET ADDRESSES.

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

[	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	Safe

#### SEE ALSO gethostbyname(3NSL), getipnodebyname(3SOCKET), getnetbyname(3SOCKET), inet(3HEAD), hosts(4), ipnodes(4), networks(4), attributes(5)

- **NOTES** The return value from inet\_ntoa() points to a buffer which is overwritten on each call. This buffer is implemented as thread-specific data in multithreaded applications.
- **BUGS** The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

inet\_addr(3XNET)

NAME	inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <arpa inet.h=""></arpa></pre>	
	in_addr_t <b>inet</b>	<b>_addr</b> (const char * <i>cp</i> );
	<pre>in_addr_t inet_lnaof(struct in_addr in);</pre>	
	<pre>struct in_addr inet_makeaddr(in_addr_t net, in_addr_t lna);</pre>	
	<pre>in_addr_t inet_netof(struct in_addr in);</pre>	
	in_addr_t <b>inet</b>	:_ <b>network</b> (const char * <i>cp</i> );
	char * <b>inet_ntc</b>	<b>pa</b> (struct in_addr <i>in</i> );
DESCRIPTION		) function converts the string pointed to by <i>cp</i> , in the Internet tion, to an integer value suitable for use as an Internet address.
	_	() function takes an Internet host address specified by <i>in</i> and network address part, in host byte order.
	The inet_makeaddr() function takes the Internet network number specified by <i>net</i> and the local network address specified by <i>lna</i> , both in host byte order, and constructs an Internet address from them.	
	The inet_netof() function takes an Internet host address specified by <i>in</i> and extracts the network number part, in host byte order.	
	The inet_network() function converts the string pointed to by <i>cp</i> , in the Internet standard dot notation, to an integer value suitable for use as an Internet network number.	
	The inet_ntoa() function converts the Internet host address specified by <i>in</i> to a string in the Internet standard dot notation.	
	All Internet addres	sses are returned in network order (bytes ordered from left to right).
	Values specified us	sing dot notation take one of the following forms:
	a.b.c.d When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.	
	a.b.c When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as 128.net.host.	
	a.b When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the rightmost three bytes of the	

	inet_addr(3XINET)			
	network address. This makes the two-part address format convenient for specifying Class A network addresses as <i>net</i> . <i>host</i> .			
		iven, the value is stored directly in the at any byte rearrangement.		
	All numbers supplied as parts in dot notation may be decimal, octal, or hexadecimal, that is, a leading 0x or 0X implies hexadecimal, as specified in the <i>ISO C</i> standard; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal.			
USAGE	The return value of inet_ntoa() may point to static data that may be overwritten by subsequent calls to inet_ntoa().			
<b>RETURN VALUES</b>	Upon successful completion, inet_addr() returns the Internet address. Otherwise, it returns (in_addr_t)(-1).			
	Upon successful completion, inet_network() returns the converted Internet network number. Otherwise, it returns (in_addr_t)(-1).			
	The inet_makeaddr() function returns the	ne constructed Internet address.		
	The inet_lnaof() function returns the local network address part.			
	The inet_netof() function returns the network number.			
	The inet_ntoa() function returns a pointer to the network address in Internet-standard dot notation.			
ERRORS	No errors are defined.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
		ATTRIBUTE VALUE		
	MT-Level MT-Safe			
SEE ALSO	<pre>endhostent(3XNET), endnetent(3XNET), attributes(5)</pre>			

# ldap(3LDAP)

NAME	ldap – Lightweight Directory Access Protocol package
SYNOPSIS	<pre>cc[ flag ] filelldap[ library ] #include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
DESCRIPTION	The Lightweight Directory Access Protocol ("LDAP") package (SUNWlldap) includes various command line LDAP clients and a LDAP client library to provide programmatic access to the LDAP protocol. This man page gives an overview of the LDAP client library functions.
	An application might use the LDAP client library functions as follows. The application would initialize a LDAP session with a LDAP server by calling ldap_init(3LDAP). Next, it authenticates to the LDAP server by calling ldap_sas1_bind(3LDAP) and friends. It may perform some LDAP operations and obtain results by calling ldap_search(3LDAP) and friends. To parse the results returned from these functions, it calls ldap_parse_result(3LDAP),ldap_next_entry(3LDAP), and ldap_first_entry (3LDAP) and others. It closes the LDAP session by calling ldap_unbind(3LDAP).
	LDAP operations can be either synchronous or asynchronous. By convention, the names of the sychronous functions end with "_s." For example, a synchronous binding to the LDAP server can be performed by calling tldap_sasl_bind_s(3LDAP). Complete an asynchronous binding with ldap_sasl_bind(3LDAP). All synchronous functions return the actual outcome of the operation, either LDAP_SUCCESS or an error code. Asynchronous routines provide an invocation identifier which can be used to obtain the result of a specific operation by passing it to theldap_result(3LDAP) function.
Initializing a LDAP session	Initializing a LDAP session involves calling the ldap_init(3LDAP) function. However, the call does not actually open a connection to the LDAP server. It merely initializes a LDAP structure that represents the session. The connection is opened when the first operation is attempted. Unlike ldap_init(), ldap_open(3LDAP) attempts to open a connection with the LDAP server. However, the use of ldap_open() is deprecated.
Authenticating to a LDAP server	The ldap_sasl_bind(3LDAP) and ldap_sasl_bind_s(3LDAP) functions provide general and extensible authenticaton for an LDAP client to a LDAP server. Both use the Simple Authentication Security Layer (SASL). Simplified routines ldap_simple_bind(3LDAP) and ldap_simple_bind_s(3LDAP) use cleartext passwords to bind to the LDAP server. Use of ldap_bind(3LDAP) and ldap_bind_s(3LDAP)(3LDAP) is deprecated.
Searching a LDAP directory	Search for an entry in a LDAP directory by calling the ldap_search_ext(3LDAP) or the ldap_search_ext_s(3LDAP) functions. These functions support LDAPv3 server controls, client controls and variable size and time limits as arguments for each search operation. ldap_search(3LDAP) and ldap_search_s(3LDAP) are identical functions but do not support the controls and limits as arguments to the call.

284 man pages section 3: Networking Library Functions • Last Revised 26 Oct 2001

Adding or Deleting an entry	Use ldap_add_ext(3LDAP) and ldap_delete_ext(3LDAP) to add or delete entries in a LDAP directory server. The synchronous counterparts to these functions are ldap_add_ext_s(3LDAP) and ldap_delete_ext_s(3LDAP). The ldap_add(3LDAP), ldap_add_s(3LDAP), ldap_delete(3LDAP), and ldap_delete_s(3LDAP) provide identical functionality to add and to delete entries, but they do not support LDAP v3 server and client controls.
Modifying Entries	Use ldap_modify_ext(3LDAP) and ldap_modify_ext_s(3LDAP) to modify an existing entry in a LDAP server that supports for LDAPv3 server and client controls. Similarly, use ldap_rename(3LDAP) and ldap_rename_s(3LDAP) to change the name of an LDAP entry. The ldap_modrdn(3LDAP), ldap_modrdn_s(), ldap_modrdn2(3LDAP) and ldap_modrdn2_s(3LDAP) interfaces are deprecated.
Obtaining Results	Use ldap_result(3LDAP) to obtain the results of a previous asynchronous operation. For all LDAP operations other than search, only one message is returned. For the search operation, a list of result messages can be returned.
Handling Errors and Parsing Results	Use the ldap_parse_result(3LDAP), ldap_parse_sasl_bind_result(3LDAP), and the ldap_parse_extended_result(3LDAP) functions to extract required information from results and and to handle the returned errors. To covert a numeric error code into a null-terminated character string message describing the error, use ldap_err2string(3LDAP). The ldap_result2error(3LDAP) and ldap_perror(3LDAP) functions are deprecated. To step through the list of messages in a result returned by ldap_result(), use ldap_first_message(3LDAP) and ldap_next_message(3LDAP). ldap_count_messages(3LDAP) returns the number of messages contained in the list. You can use ldap_first_entry(3LDAP) and ldap_next_entry(3LDAP) to step through and obtain a list of entries from a list of messages returned by a search result. ldap_count_entries(3LDAP) returns the number of entries contained in a list of messages. Call either ldap_first_attribute(3LDAP) and ldap_next_attribute(3LDAP) to step through a list of attributes associated with an entry. Retrieve the values of a given attribute by calling ldap_get_values(3LDAP) and ldap_get_values(3LDAP). Count the number of values returned by using ldap_count_values(3LDAP) and ldap_count_values_len(3LDAP). Use the ldap_get_lang_values(3LDAP) and ldap_get_lang_values_len(3LDAP) to return an attribute's values that matches a specified language subtype. The ldap_get_lang_values() function returns an array of an attribute's string values that matches a specified language subtype. To retrieve the binary data from an attribute, call the ldap_get_lang_values_len() function instead.

### ldap(3LDAP)

Uniform Resource Locators (URLS)	You can use the ldap_url(3LDAP)functions to test a URL to verify that it is an LDAP URL, to parse LDAP URLs into their component pieces, to initiate searches directly using an LDAP URL, and to retrieve the URL associated with a DNS domain name or a distinguished name.
User Friendly Naming	The ldap_ufn(3LDAP) functions implement a user friendly naming scheme by means of LDAP. This scheme allows you to look up entries using fuzzy, untyped names like "mark smith, umich, us".
Caching	The ldap_memcache(3LDAP) functions provide an in-memory client side cache to store search requests. Caching improves performance and reduces network bandwidth when a client makes repeated requests.
Utility Functions	There are also various utility functions. You can use the ldap_sort(3LDAP) functions are used to sort the entries and values returned by means of the ldap search functions. The ldap_friendly(3LDAP) functions will map from short two letter country codes or other strings to longer "friendlier" names. Use the ldap_charset(3LDAP) functions to translate to and from the T.61 character set that is used for many character strings in the LDAP protocol.
Generating Filters	Make calls to ldap_init_getfilter(3LDAP) and ldap_init_getfilter_buf() to generate filters to be used in ldap_search(3LDAP) and ldap_search_s(3LDAP).ldap_init_getfilter() reads ldapfilter.conf(4), the LDAP configuration file, while ldap_init_getfilter_buf() reads the configuration information from <i>buf</i> of length <i>buflen</i> .ldap_getfilter_free(3LDAP) frees memory that has been allocated by means of ldap_init_getfilter().
BER Library	The LDAP package includes a set of lightweight Basic Encoding Rules ("BER)" functions. The LDAP library functions use the BER functions to encode and decode LDAP protocol elements through the slightly simplified BER defined by LDAP. They are not normally used directly by An LDAP application program will not normally use the BER functions directly. Instead, these functions provide a printf() and scanf()-like interface, as well as lower-level access.
LIST OF INTERFACES	ldap_open(3LDAP) Deprecated. Use ldap_init(3LDAP).
	<pre>ldap_init(3LDAP) Initialize a session with a LDAP server without opening a connection to a server.</pre>
	ldap_result(3LDAP) Obtain the result from a previous asynchronous operation.
	ldap_abandon(3LDAP) Abandon or abort an asynchronous operation.
	ldap_add(3LDAP) Asynchronously add an entry
	ldap_add_s(3LDAP) Synchronously add an entry.

<sup>286</sup> man pages section 3: Networking Library Functions • Last Revised 26 Oct 2001

http(0DD11)
<pre>ldap_add_ext(3LDAP)    Asynchronously add an entry with support for LDAPv3 controls.</pre>
ldap_add_ext_s(3LDAP) Synchronously add an entry with support for LDAPv3 controls.
<pre>ldap_bind(3LDAP)     Deprecated. Use ldap_sasl_bind(3LDAP) or ldap_simple_bind(3LDAP).</pre>
ldap_sasl_bind(3LDAP) Asynchronously bind to the directory using SASL authentication
ldap_sasl_bind_s(3LDAP) Synchronously bind to the directory using SASL authentication
<pre>ldap_bind_s(3LDAP) Deprecated. Use ldap_sasl_bind_s(3LDAP) or ldap_simple_bind_s(3LDAP).</pre>
<pre>ldap_simple_bind(3LDAP) Asynchronously bind to the directory using simple authentication.</pre>
<pre>ldap_simple_bind_s(3LDAP) Synchronously bind to the directory using simple authentication.</pre>
<pre>ldap_unbind(3LDAP) Synchronously unbind from the LDAP server, close the connection, and dispose the session handle.</pre>
<pre>ldap_unbind_ext(3LDAP)     Synchronously unbind from the LDAP server and close the connection.     ldap_unbind_ext() allows you to explicitly include both server and client     controls in the unbind request.</pre>
<pre>ldap_set_rebind_proc(3LDAP) Set callback function for obtaining credentials from a referral.</pre>
<pre>ldap_memcache_init(3LDAP) Create the in-memory client side cache.</pre>
<pre>ldap_memcache_set(3LDAP) Associate an in-memory cache that has been already created by calling the ldap_memcache_init(3LDAP) function with an LDAP connection handle.</pre>
ldap_memcache_get(3LDAP) Get the cache associated with the specified LDAP structure.
ldap_memcache_flush(3LDAP) Flushes search requests from the cache.
ldap_memcache_destroy(3LDAP) Frees the specified LDAPMemCache structure pointed to by cache from memory.
<pre>ldap_memcache_update(3LDAP)     Checks the cache for items that have expired and removes them.</pre>

# ldap(3LDAP)

ldap_compare(3LDAP) Asynchronous compare with a directory entry.
ldap_compare_s(3LDAP) Synchronous compare with a directory entry.
ldap_compare_ext(3LDAP) Asynchronous compare with a directory entry, with support for LDAPv3 controls
<pre>ldap_compare_ext_s(3LDAP) Synchronous compare with a directory entry, with support for LDAPv3 controls.</pre>
ldap_control_free(3LDAP) Dispose of an LDAP control.
ldap_controls_free(3LDAP) Dispose of an array of LDAP controls.
ldap_delete(3LDAP) Asynchronously delete an entry.
ldap_delete_s(3LDAP) Synchronously delete an entry.
<pre>ldap_delete_ext(3LDAP) Asynchronously delete an entry, with support for LDAPv3 controls.</pre>
<pre>ldap_delete_ext_s(3LDAP) Synchronously delete an entry, with support for LDAPv3 controls.</pre>
<pre>ldap_init_templates(3LDAP)     Read a sequence of templates from a LDAP template configuration file.</pre>
<pre>ldap_init_templates_buf(3LDAP)     Read a sequence of templates from a buffer.</pre>
<pre>ldap_free_templates(3LDAP) Dispose of the templates allocated.</pre>
<pre>ldap_first_reference(3LDAP) Step through a list of continuation references from a search result.</pre>
<pre>ldap_next_reference(3LDAP) Step through a list of continuation references from a search result.</pre>
ldap_count_references(3LDAP) Count the number of messages in a search result.
ldap_first_message(3LDAP) Step through a list of messages in a search result.
ldap_count_messages(3LDAP) Count the messages in a list of messages in a search result.
ldap_next_message(3LDAP) Step through a list of messages in a search result.

ldap_msgtype(3LDAP) Return the type of LDAP message.
ldap_first_disptmpl(3LDAP) Get first display template in a list.
ldap_next_disptmpl(3LDAP) Get next display template in a list.
ldap_oc2template(3LDAP) Return template appropriate for the objectclass.
ldap_name2template(3LDAP) Return named template
ldap_tmplattrs(3LDAP) Return attributes needed by the template.
<pre>ldap_first_tmplrow(3LDAP)     Return first row of displayable items in a template.</pre>
<pre>ldap_next_tmplrow(3LDAP)     Return next row of displayable items in a template.</pre>
ldap_first_tmplcol(3LDAP) Return first column of displayable items in a template.
<pre>ldap_next_tmplcol(3LDAP)     Return next column of displayable items in a template.</pre>
ldap_entry2text(3LDAP) Display an entry as text by using a display template.
<pre>ldap_entry2text_search(3LDAP) Search for and display an entry as text by using a display template.</pre>
ldap_vals2text(3LDAP) Display values as text.
<pre>ldap_entry2html(3LDAP) Display an entry as HTML (HyperText Markup Language) by using a display template.</pre>
ldap_entry2html_search(3LDAP) Search for and display an entry as HTML by using a display template.
ldap_vals2html(3LDAP) Display values as HTML.
<pre>ldap_perror(3LDAP) Deprecated. Use ldap_parse_result(3LDAP).</pre>
<pre>ldap_result2error(3LDAP)     Deprecated. Use ldap_parse_result(3LDAP).</pre>

## ldap(3LDAP)

<pre>ldap_errlist(3LDAP) List ldap errors and their meanings.</pre>
<pre>ldap_err2string(3LDAP) Convert LDAP error indication to a string.</pre>
ldap_first_attribute(3LDAP) Return first attribute name in an entry.
ldap_next_attribute(3LDAP) Return next attribute name in an entry.
ldap_first_entry(3LDAP) Return first entry in a chain of search results.
ldap_next_entry(3LDAP) Return next entry in a chain of search results.
ldap_count_entries(3LDAP) Return number of entries in a search result.
<pre>ldap_friendly_name(3LDAP) Map from unfriendly to friendly names.</pre>
<pre>ldap_free_friendlymap(3LDAP) Free resources used by ldap_friendly(3LDAP).</pre>
ldap_get_dn(3LDAP) Extract the DN from an entry.
ldap_explode_dn(3LDAP) Convert a DN into its component parts.
ldap_explode_dns(3LDAP) Convert a DNS-style DN into its component parts (experimental).
ldap_is_dns_dn(3LDAP) Check to see if a DN is a DNS-style DN (experimental).
ldap_dns_to_dn(3LDAP) Convert a DNS domain name into an X.500 distinguished name.
ldap_dn2ufn(3LDAP) Convert a DN into user friendly form.
ldap_get_values(3LDAP) Return an attribute's values.
<pre>ldap_get_values_len(3LDAP)     Return an attribute's values with lengths.</pre>
<pre>ldap_value_free(3LDAP) Free memory allocated by ldap_get_values(3LDAP).</pre>
<pre>ldap_value_free_len(3LDAP) Free memory allocated by ldap_get_values_len(3LDAP).</pre>

# ldap(3LDAP)

	rearp
ldap_count_values(3LDAP) Return number of values.	
<pre>ldap_count_values_len(3LDAP)     Return number of values.</pre>	
<pre>ldap_init_getfilter(3LDAP) Initialize getfilter functions from a file.</pre>	
<pre>ldap_init_getfilter_buf(3LDAP) Initialize getfilter functions from a buffer.</pre>	
<pre>ldap_getfilter_free(3LDAP) Free resources allocated by ldap_init_getfilter(3LDAP).</pre>	
ldap_getfirstfilter(3LDAP) Return first search filter.	
ldap_getnextfilter(3LDAP) Return next search filter.	
ldap_build_filter(3LDAP) Construct an LDAP search filter from a pattern.	
<pre>ldap_setfilteraffixes(3LDAP) Set prefix and suffix for search filters.</pre>	
ldap_modify(3LDAP) Asynchronously modify an entry.	
ldap_modify_s(3LDAP) Synchronously modify an entry.	
<pre>ldap_modify_ext(3LDAP) Asynchronously modify an entry, return value, and place message.</pre>	
<pre>ldap_modify_ext_s(3LDAP) Synchronously modify an entry, return value, and place message.</pre>	
<pre>ldap_mods_free(3LDAP) Free array of pointers to mod structures used by ldap_modify(3LD</pre>	DAP).
ldap_modrdn2(3LDAP) Deprecated. Use ldap_rename(3LDAP) instead.	
ldap_modrdn2_s(3LDAP) Deprecated. Use ldap_rename_s(3LDAP) instead.	
ldap_modrdn(3LDAP) Deprecated. Use ldap_rename(3LDAP) instead.	
ldap_modrdn_s(3LDAP) Depreciated. Use ldap_rename_s(3LDAP) instead.	
ldap_rename(3LDAP) Asynchronously modify the name of an LDAP entry.	

## ldap(3LDAP)

Idap_rename_s(3LDAP) Synchronously modify the name of an LDAP entry.
ldap_msgfree(3LDAP) Free result messages.
ldap_parse_result(3LDAP) Search for a message to parse.
<pre>ldap_parse_extended_result(3LDAP)    Search for a message to parse.</pre>
<pre>ldap_parse_sasl_bind_result(3LDAP) Search for a message to parse.</pre>
ldap_search(3LDAP) Asynchronously search the directory.
ldap_search_s(3LDAP) Synchronously search the directory.
ldap_search_ext(3LDAP) Asynchronously search the directory with support for LDAPv3 controls.
<pre>ldap_search_ext_s(3LDAP) Synchronously search the directory with support for LDAPv3 controls.</pre>
ldap_search_st(3LDAP) Synchronously search the directory with support for a local timeout value.
ldap_ufn_search_s(3LDAP) User friendly search the directory.
ldap_ufn_search_c(3LDAP) User friendly search the directory with cancel.
ldap_ufn_search_ct(3LDAP) User friendly search the directory with cancel and timeout.
<pre>ldap_ufn_setfilter(3LDAP) Set filter file used by ldap_ufn(3LDAP) functions.</pre>
<pre>ldap_ufn_setprefix(3LDAP) Set prefix used by ldap_ufn(3LDAP) functions.</pre>
<pre>ldap_ufn_timeout(3LDAP) Set timeout used by ldap_ufn(3LDAP) functions.</pre>
ldap_is_ldap_url(3LDAP) Check a URL string to see if it is an LDAP URL.
ldap_url_parse(3LDAP) Break up an LDAP URL string into its components.
ldap_free_urldesc(3LDAP) Free an LDAP URL structure.

ldap_url_search(3LDAP) Asynchronously search by using an LDAP URL.
ldap_url_search_s(3LDAP) Synchronously search by using an LDAP URL.
<pre>ldap_url_search_st(3LDAP)    Asynchronously search by using an LDAP URL, with support for a local timeout    value.</pre>
ldap_dns_to_url(3LDAP) Locate the LDAP URL associated with a DNS domain name.
ldap_dn_to_ur1(3LDAP) Locate the LDAP URL associated with a distinguished name.
<pre>ldap_init_searchprefs(3LDAP) Initialize searchprefs functions from a file.</pre>
<pre>ldap_init_searchprefs_buf(3LDAP) Initialize searchprefs functions from a buffer.</pre>
<pre>ldap_free_searchprefs(3LDAP) Free memory allocated by searchprefs functions.</pre>
ldap_first_searchobj(3LDAP) Return first searchpref object.
ldap_next_searchobj(3LDAP) Return next searchpref object.
ldap_sort_entries(3LDAP) Sort a list of search results.
ldap_sort_values(3LDAP) Sort a list of attribute values.
ldap_sort_strcasecmp(3LDAP) Case insensitive string comparison.
<pre>ldap_set_string_translators(3LDAP) Set character set translation functions used by LDAP library.</pre>
ldap_translate_from_t61(3LDAP) Translate from the T.61 character set to another character set.
ldap_translate_to_t61(3LDAP) Translate to the T.61 character set from another character set.
ldap_enable_translation(3LDAP) Enable or disable character translation for an LDAP entry result.
ldap_version(3LDAP) Get version information about the LDAP SDK for C.

ldap(3LDAP)	
	ldap_get_lang_values(3LDAP) Return an attribute's value that matches a specified language subtype.
	<pre>ldap_get_lang_values_len(3LDAP)     Return an attribute's value that matches a specified language subtype along with     lengths.</pre>
	ldap_get_entry_controls(3LDAP) Get the LDAP controls included with a directory entry in a set of search results.
	ldap_get_option(3LDAP) Get session preferences in an LDAP structure.
	ldap_set_option(3LDAP) Set session preferences in an LDAP structure.
ATTRIBUTES	See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Stability Level	Evolving

### **SEE ALSO** attributes(5)

		( )
NAME	ldap_abandon – abandon an LDAP operation in progress	
SYNOPSIS	cc[ flag ] filelldap[ library ]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	int <b>ldap_abandon</b> (LDAP *ld, int msg	gid);
<b>DESCRIPTION</b> The ldap_abandon() function is used to abandon or cancel an LDAP opprogress. The <i>msgid</i> passed should be the message id of an outstanding LD operation, as returned by ldap_search(3LDAP), ldap_modify(3LDAP)		nessage id of an outstanding LDAP
	ldap_abandon() checks to see if the result has, it deletes it from the queue of pending abandon operation to the the LDAP server.	
	The caller can expect that the result of an a from a future call to ldap_result(3LDA)	
ERRORS	ldap_abandon() returns 0 if successful c appropriately. See ldap_error(3LDAP) for	8
ATTRIBUTES	See attributes(5) for a description of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWIIdap (32-bit)
		SUNWldapx (64-bit)
	Stability Level	Evolving
SEE ALSO	ldap(3N), ldap_result(3N), ldap_e	

## ldap\_add(3LDAP)

NAME	ldap_add, ldap_add_s, ldap_add_ext, ldap_add_ext_s – perform an LDAP add operation	
SYNOPSIS	cc[flag] filelldap[ library]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	int <b>ldap_add</b> (LDAP *ld, char *dn, LDA	APMod *attrs[]);
	int <b>ldap_add_s</b> (LDAP *ld, char *dn, I	DAPMod *attrs[]);
	<pre>int ldap_add_ext(LDAP *ld, char *dr</pre>	1, LDAPMod **attrs, LDAPControl
	<pre>int ldap_add_ext_s(LDAP *ld, char     **serverctrls, LDAPControl **clientce</pre>	
DESCRIPTION	The ldap_add_s() function is used to per the DN of the entry to add, and <i>attrs</i> , a null The LDAPMod structure is used to represen <i>mod_values</i> fields being used as described us <i>ldap_op</i> field being used only if you need to Otherwise, it should be set to zero.	-terminated array of the entry's attributes. nt attributes, with the <i>mod_type</i> and nder ldap_modify(3LDAP), and the
	Note that all entries except that specified by already exist. ldap_add_s() returns an Ll failure of the operation. See ldap_error(3	DAP error code indicating success or
	The ldap_add() function works just like I returns the message id of the request it initi obtained by calling ldap_result(3LDAP)	ated. The result of this operation can be
	The ldap_add_ext() function initiates an LDAP_SUCCESS if the request was successful LDAP error code if not (see ldap_error(3 places the message id of *msgidp. A sub used to obtain the result of the add request.	ully sent to the server, or else it returns a LDAP)). If successful, ldap_add_ext() bsequent call to ldap_result(), can be
	The ldap_add_ext_s() function initiates the result of the operation itself.	a synchronous add operation and returns
ERRORS	field in the ld parameter to indicate the error. ldap_add_s() will return an LDAP error code directly (LDAP_SUCCESS if everything went ok, an error otherwise).	
ATTRIBUTES		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE

## ldap\_add(3LDAP)

Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

## **SEE ALSO** ldap(3LDAP), ldap\_error(3LDAP), ldap\_modify(3LDAP)

## ldap\_ber\_free(3LDAP)

NAME	ldap_ber_free – free a BerElement structure from memory	
SYNOPSIScc -flag file lldap [ -library ] #include <ldap.h>void ldap_ber_free(BerElement *ber, int freebuf);</ldap.h>		prary ]
		nt *ber, int freebuf);
DESCRIPTION	You can make a call to the ldap_ber_free() function to free BerElement structures allocated by ldap_first_attribute() and by ldap_next_attribute() function calls. When freeing structures allocated by these functions, specify 0 for the <i>freebuf</i> argument. The ldap_first_attribute() and by ldap_next_attribute() functions do not allocate the extra buffer in the BerElement structure.	
	For example, to retrieve attributes from a search result entry, you need to call the ldap_first_attribute() function. A call to this function allocates a BerElement structure, which is used to help track the current attribute. When you are done working with the attributes, this structure should be freed from memory, if it still exists.	
	This function is deprecated . Use the	ber_free() function instead.
ATTRIBUTES	See attributes(5) for descriptions	of the following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWcsl (32-bit)
		SUNWcslx (64–bit)
	Interface Stability	SUNWcslx (64-bit) Obsolete

NAME	ldap_bind, ldap_bind_s, ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, ldap_simple_bind_s, ldap_unbind, ldap_unbind_s, ldap_unbind_ext, ldap_set_rebind_proc – LDAP bind functions
SYNOPSIS	cc [ flag ] filelldap[ library ]
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
	<pre>int ldap_bind(LDAP *ld, char *who, char *cred, int method);</pre>
	<pre>int ldap_bind_s(LDAP *ld, char *who, char *cred, int method);</pre>
	<pre>int ldap_simple_bind(LDAP *ld, char *who, char *passwd);</pre>
	<pre>int ldap_simple_bind_s(LDAP *ld, char *who, char *passwd);</pre>
	<pre>int ldap_unbind(LDAP *ld);</pre>
	<pre>int ldap_unbind_s(LDAP *ld);</pre>
	<pre>int ldap_unbind_ext(LDAP *ld, LDAPControl **serverctrls, LDAPControl</pre>
	<pre>void ldap_set_rebind_proc(LDAP *ld, int (*rebindproc);</pre>
	<pre>int ldap_sasl_bind(LDAP *ld, char *dn, char *mechanism, struct berval **serverctrls, LDAPControl **clientctrls, int *msgidp);</pre>
	<pre>int ldap_sasl_bind_s(LDAP *ld, char *dn, char *mechanism, struct berval *cred, LDAPControl **serverctrls, LDAPControl **clientctrls);</pre>
DESCRIPTION	These functions provide various interfaces to the LDAP bind operation. After a connection is made to an LDAP server using ldap_open(3LDAP), an LDAP bind operation must be performed before other operations can be attempted over the conection. Both synchronous and asynchronous versions of each variant of the bind call are provided. There are three types of bind calls, simple authentication, kerberos authentication, and general functions. All LDAP bind functions take <i>ld</i> as their first parameter, which is returned from ldap_open(3LDAP).
Simple Authentication	The simplest form of the bind call is ldap_simple_bind_s(). It takes the DN to bind as in who, and the userPassword associated with the entry in <i>passwd</i> . It returns an LDAP error code. See ldap_error(3LDAP). The ldap_simple_bind() call is asynchronous. It takes the same parameters but only initiates the bind operation and returns the message id of the request it sent. You can obtain the result of the operation by a subsequent call to ldap_result(3LDAP).
General Authentication	Use the ldap_bind() and ldap_bind_s() functions when the authentication method to use needs to be selected at runtime. Both functions take an extra <i>method</i> parameter that selects the authentication method to use. It should be set to LDAP_AUTH_SIMPLE to select simple authentication.ldap_bind() returns the message id of the request it initiates.ldap_bind_s() returns an LDAP error code.

# ldap\_bind(3LDAP)

Unbinding	The ldap_sasl_bind() and ldap_sasl_bind_s() functions are used for general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer. The routines both take the dn to bind as the method to use. A dotted-string representation of an OID identifies the method, and a struct berval holds the credentials. The special constant value LDAP_SASL_SIMPLE ("") can be passed to request simple authentication, or the simplified routines ldap_simple_bind() or ldap_simple_bind_s() can be use. The ldap_unbind() call is used to unbind from the directory, terminate the current association, and free the resources contained in the <i>ld</i> structure. Once it is called, the connection to the LDAP server is closed, and the <i>ld</i> structure is invalid. The ldap_unbind_s() call is just another name for ldap_unbind(). Both of these calls
	are synchronous in nature. The ldap_unbind_ext() function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure. Unlike ldap_unbind() and ldap_unbind_s(), you can explicitly include both server and client controls with the with ldap_unbind_ext() request. Since there is no server reponse to an unbind request, you will not receive a response from a server control that is included with the unbind request.
Rebinding While Following Referral	The ldap_set_rebind_proc() call is used to set a function that will be called back to obtain bind credentials used when a new server is contacted following an LDAP referral. If ldap_set_rebind_proc() is never called, or if it is called with a NULL <i>rebindproc</i> parameter, an unauthenticated simple LDAP bind will always be done when chasing referrals.
	rebindproc() should be a function that is declared like this:
	<pre>int rebindproc(LDAP *ld, char **whop, char **credp, int *methodp, int freeit);</pre>
	The LDAP library will first call the rebindproc() to obtain the referral bind credentials, and the <i>freeit</i> parameter will be zero. The <i>whop</i> , <i>credp</i> , and <i>methodp</i> should be set as appropriate. If rebindproc() returns LDAP_SUCCESS, referral processing continues, and the rebindproc() will be called a second time with <i>freeit</i> non-zero to give your application a chance to free any memory allocated in the previous call.
	If anything but LDAP_SUCCESS is returned by the first call to rebindproc(), then referral processing is stopped, and that error code is returned for the original LDAP operation.
<b>RETURN VALUES</b>	Make a call to ldap_result(3LDAP) to obtain the result of a bind operation.
ERRORS	Asynchronous functions will return -1 in case of error. See ldap_error(3LDAP) for more information on error codes returned If no credentials are returned, the result parameter is set to NULL.
ATTRIBUTES	See attributes(5) for a description of the following attributes:

300 man pages section 3: Networking Library Functions • Last Revised 25 Oct 2001

ldap\_bind(3LDAP)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Stability Level	Evolving

**SEE ALSO** ldap(3LDAP), ldap\_error(3LDAP), ldap\_open(3LDAP), attributes(5)

# ldap\_charset(3LDAP)

NAME	E ldap_charset, ldap_set_string_translators, ldap_t61_to_8859, ldap_8859_to_t61, ldap_translate_from_t61, ldap_translate_to_t61, ldap_enable_translation – LDAP character set translation functions	
SYNOPSIS	cc[flag] filelldap[ library]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>void ldap_set_string_translators(LDAP *ld, BERTranslateProc encode_proc, BERTranslateProc decodeproc);</pre>	
	<pre>typedef int(*BERTranslateProc)(char **bufp, unsigned long *buflen int free_input);</pre>	
	<pre>int ldap_t61_to_8859(char **bufp, unsigned long *buflenp, int     free_input);</pre>	
	<pre>int ldap_8859_to_t61(char **bufp, unsigned long *buflenp, int     free_input);</pre>	
	<pre>int ldap_translate_from_t61(LDAP *ld, char **bufp, unsigned long</pre>	
	<pre>int ldap_translate_to_t61(LDAP *ld, char **bufp, unsigned long *lenp,</pre>	
	<pre>void ldap_enable_translation(LDAP *ld, LDAPMessage *entry, int enable);</pre>	
DESCRIPTION	These functions are used to used to enable translation of character strings used in the LDAP library to and from the T.61 character set used in the LDAP protocol. These functions are only available if the LDAP and LBER libraries are compiled with STR_TRANSLATION defined. It is also possible to turn on character translation by default so that all LDAP library callers will experience translation; see the LDAP Make-common source file for details.	
	ldap_set_string_translators() sets the translation functions that will be used by the LDAP library. They are not actually used until the <i>ld_lberoptions</i> field of the LDAP structure is set to include the LBER_TRANSLATE_STRINGS option.	
	<pre>ldap_t61_to_8859() and ldap_8859_to_t61() are translation functions for converting between T.61 characters and ISO-8859 characters. The specific 8859 character set used is determined at compile time.</pre>	
	<pre>ldap_translate_from_t61() is used to translate a string of characters from the T.61 character set to a different character set. The actual translation is done using the decode_proc that was passed to a previous call to ldap_set_string_translators(). On entry, *bufp should point to the start of the T.61 characters to be translated and *lenp should contain the number of bytes to translate. If free_input is non-zero, the input buffer will be freed if translation is a success. If the translation is a success, LDAP_SUCCESS will be returned, *bufp will</pre>	

point to a newly malloc'd buffer that contains the translated characters, and *\*lenp* will contain the length of the result. If translation fails, an LDAP error code will be returned.

ldap\_translate\_to\_t61() is used to translate a string of characters to the T.61 character set from a different character set. The actual translation is done using the encode\_proc that was passed to a previous call to ldap\_set\_string\_translators(). This function is called just like ldap\_translate from t61().

ldap\_enable\_translation() is used to turn on or off string translation for the LDAP entry entry (typically obtained by calling ldap\_first\_entry() or ldap\_next\_entry() after a successful LDAP search operation). If enable is zero, translation is disabled; if non-zero, translation is enabled. This function is useful if you need to ensure that a particular attribute is not translated when it is extracted using ldap\_get\_values() or ldap\_get\_values\_len(). For example, you would not want to translate a binary attributes such as jpegPhoto.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWlldap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

### SEE ALSO | 1dap(3LDAP)

## ldap\_compare(3LDAP)

NAME	ldap_compare, ldap_compare_s, ldap_compare_ext, ldap_compare_ext_s – LDAP compare operation		
SYNOPSIS	cc[flag] file11dap[library]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>int ldap_compare(LDAP *ld, char *dn, char *attr, char *value);</pre>		
	<pre>int ldap_compare_s(LDAP *ld, char *dn, char *attr, char *value);</pre>		
	<pre>int ldap_compare_ext(LDAP *ld, char *dn, char *attr, struct berval</pre>		
	<pre>int ldap_compare_ext_s(LDAP *ld, c  *bvalue, LDAPControl **serverctrls,</pre>		
DESCRIPTION	The ldap_compare_s() function is used synchronously. It takes <i>dn</i> , the DN of the er and <i>attr</i> and <i>value</i> , the attribute type and va It returns an LDAP error code, which will b contains the attribute value and LDAP_COM some error code is returned.	ntry upon which to perform the compare, alue to compare to those found in the entry. De LDAP_COMPARE_TRUE if the entry	
	The ldap_compare() function is used to perform an LDAP compare operation asynchronously. It takes the same parameters as ldap_compare_s(), but returns the message id of the request it initiated. The result of the compare can be obtained by a subsequent call to ldap_result(3LDAP).		
	The ldap_compare_ext() function initiates an asynchronous compare operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code if not (see ldap_error(3LDAP). If successful, ldap_compare_ext() places the message id of the request in * <i>msgidp</i> . A subsequent call to ldap_result(), can be used to obtain the result of the add request.		
	The ldap_compare_ext_s() function initiates a synchronous compare operation and as such returns the result of the operation itself.		
ERRORS	<pre>ldap_compare_s() returns an LDAP error code which can be interpreted by calling one of ldap_perror(3LDAP) and friends. ldap_compare() returns -1 if something went wrong initiating the request. It returns the non-negative message id of the request if it was successful.</pre>		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	

304 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

ldap\_compare(3LDAP)

	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** | ldap(3LDAP), ldap\_error(3LDAP)

**BUGS** There is no way to compare binary values but there should be.

# ldap\_control\_free(3LDAP)

NAME	ldap_control_free, ldap_controls_free – LDAP control disposal	
SYNOPSIS	cc[flag] filelldap[ library]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>void ldap_control_free(LDAPControl *ctrl);</pre>	
	<pre>void ldap_controls_free(LDAPControl *ctrls);</pre>	
DESCRIPTION	<pre>ldap_controls_free() and ldap_control_free() are routines which can be used to dispose of a single control or an array of controls allocated by other LDAP APIs.</pre>	
<b>RETURN VALUES</b>	None.	
ERRORS	No errors are defined for these functions.	
ATTRIBUTES	See attributes(5) for a description of the following attributes:	

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIldap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** | ldap\_error(3LDAP), ldap\_result(3LDAP), attributes(5)

NAME	ldap_delete, ldap_delete_s, ldap_delete_ext, ldap_delete_ext_s – LDAP delete operation		
SYNOPSIS	cc[flag] filelldap[library]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>int ldap_delete(LDAP *ld, char *dn);</pre>		
	<pre>int ldap_delete_s(LDAP *ld, char *dn);</pre>		
	<pre>int ldap_delete_ext(LDAP *ld, char *dn, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp);</pre>		
	<pre>int ldap_delete_ext_s(LDAP *ld, ch LDAPControl **clientctrls);</pre>	<pre>nar *dn, LDAPControl **serverctrls,</pre>	
DESCRIPTION	The ldap_delete_s() function is used to perform an LDAP delete operation synchronously. It takes <i>dn</i> , the DN of the entry to be deleted. It returns an LDAP error code, indicating the success or failure of the operation.		
	The ldap_delete() function is used to perform an LDAP delete operation asynchronously. It takes the same parameters as ldap_delete_s(), but returns the message id of the request it initiated. The result of the delete can be obtained by a subsequent call to ldap_result(3LDAP).		
	The ldap_delete_ext() function initiate returns LDAP_SUCCESS if the request was a returns a LDAP error code if not (see ldap ldap_delete_ext() places the message call to ldap_result(), can be used to obt	successfully sent to the server, or else it _error(3LDAP)). If successful, id of the request in * <i>msgidp</i> . A subsequent	
	The ldap_delete_ext_s() function initiates a synchronous delete operation and as such returns the result of the operation itself.		
ERRORS	<pre>ldap_delete_s() returns an LDAP error code which can be interpreted by calling one of ldap_perror(3LDAP) functions. ldap_delete() returns -1 if something went wrong initiating the request. It returns the non-negative message id of the request if things were successful.</pre>		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	

Stability Level

Evolving

ldap\_delete(3LDAP)

**SEE ALSO** | ldap(3LDAP), ldap\_error(3LDAP)

308 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

NAME	E   ldap_disptmpl, ldap_init_templates, ldap_init_templates_buf, ldap_free_templates,   ldap_first_disptmpl, ldap_next_disptmpl, ldap_oc2template, ldap_name2template,   ldap_tmplattrs, ldap_first_tmplrow, ldap_next_tmplrow, ldap_first_tmplcol,   ldap_next_tmplcol – LDAP display template functions	
SYNOPSIS	cc[ flag ] filelldap[ library ]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>int ldap_init_templates(char *file, struct ldap_disptmpl **tmpllistp);</pre>	
	<pre>int ldap_init_templates_buf(char *buf, unsigned long len, struct     ldap_disptmpl **tmpllistp);</pre>	
	<pre>void ldap_free_templates(struct ldap_disptmpl *tmpllist);</pre>	
	<pre>struct ldap_disptmpl *ldap_first_disptmpl(struct ldap_disptmpl</pre>	
	<pre>struct ldap_disptmpl *ldap_next_disptmpl(struct ldap_disptmpl</pre>	
	<pre>struct ldap_disptmpl *ldap_oc2template(char **oclist, struct     ldap_disptmpl *tmpllist);</pre>	
	<pre>struct ldap_disptmpl *ldap_name2template(char *name, struct ldap_disptmpl *tmpllist);</pre>	
	<pre>char **ldap_tmplattrs(struct ldap_disptmpl *tmpl, char **includeattrs,</pre>	
	<pre>struct ldap_tmplitem *ldap_first_tmplrow(struct ldap_disptmpl</pre>	
	<pre>struct ldap_tmplitem *ldap_next_tmplrow(struct ldap_disptmpl</pre>	
	<pre>struct ldap_tmplitem *ldap_first_tmplcol(struct ldap_disptmpl</pre>	
	<pre>struct ldap_tmplitem *ldap_next_tmplcol(struct ldap_disptmpl</pre>	
DESCRIPTION	These functions provide a standard way to access LDAP entry display templates. Entry display templates provide a standard way for LDAP applications to display directory entries. The general idea is that it is possible to map the list of object class values present in an entry to an appropriate display template. Display templates are defined in a configuration file (see ldaptemplates.conf(4)). Each display template contains a pre-determined list of items, where each item generally corresponds to an attribute to be displayed. The items contain information and flags that the caller can use to display the attribute and values in a reasonable fashion. Each item has a	

syntaxid, which are described in the SYNTAX IDS section below. The ldap\_entry2text(3LDAP) functions use the display template functions and produce text output.

ldap\_init\_templates() reads a sequence of templates from a valid LDAP template configuration file (see ldaptemplates.conf(4)). Upon success, 0 is returned, and *tmpllistp* is set to point to a list of templates. Each member of the list is an ldap\_disptmpl structure (defined below in the DISPTMPL Structure Elements section).

ldap\_init\_templates\_buf() reads a sequence of templates from buf (whose size is buflen). buf should point to the data in the format defined for an LDAP template configuration file (see ldaptemplates.conf(4)). Upon success, 0 is returned, and tmpllistp is set to point to a list of templates.

The LDAP\_SET\_DISPTMPL\_APPDATA() macro is used to set the value of the dt\_appdata field in an ldap\_disptmpl structure. This field is reserved for the calling application to use; it is not used internally.

The LDAP\_GET\_DISPTMPL\_APPDATA() macro is used to retrieve the value in the dt\_appdata field.

The LDAP\_IS\_DISPTMPL\_OPTION\_SET() macro is used to test a ldap\_disptmpl structure for the existence of a template option. The options currently defined are: LDAP\_DTMPL\_OPT\_ADDABLE (it is appropriate to allow entries of this type to be added), LDAP\_DTMPL\_OPT\_ALLOWMODRDN (it is appropriate to offer the "modify rdn" operation), LDAP\_DTMPL\_OPT\_ALTVIEW (this template is merely an alternate view of another template, typically used for templates pointed to be an LDAP\_SYN\_LINKACTION item).

ldap\_free\_templates() disposes of the templates allocated by
ldap init templates().

ldap\_first\_disptmpl() returns the first template in the list tmpllist. The tmpllist is
typically obtained by calling ldap\_init\_templates() .

ldap\_next\_disptmpl() returns the template after tmpl in the template list tmpllist.
A NULL pointer is returned if tmpl is the last template in the list.

ldap\_oc2template() searches tmpllist for the best template to use to display an entry that has a specific set of objectClass values. oclist should be a null-terminated array of strings that contains the values of the objectClass attribute of the entry. A pointer to the first template where all of the object classes listed in one of the template's dt\_oclist elements are contained in oclist is returned. A NULL pointer is returned if no appropriate template is found.

ldap\_tmplattrs() returns a null-terminated array that contains the names of attributes that need to be retrieved if the template *tmpl* is to be used to display an entry. The attribute list should be freed using ldap\_value\_free(). The *includeattrs* 

parameter contains a null-terminated array of attributes that should always be included (it may be NULL if no extra attributes are required). If *syntaxmask* is non-zero, it is used to restrict the attribute set returned. If *exclude* is zero, only attributes where the logical AND of the template item syntax id and the *syntaxmask* is non-zero are included. If *exclude* is non-zero, attributes where the logical AND of the template item syntax id and the *syntaxmask* is non-zero are excluded.

ldap\_first\_tmplrow() returns a pointer to the first row of items in template tmpl.

ldap\_next\_tmplrow() returns a pointer to the row that follows row in template
tmpl.

ldap\_first\_tmplcol() returns a pointer to the first item (in the first column) of row row within template tmpl. A pointer to an ldap\_tmplitem structure (defined below in the TMPLITEM Structure Elements section) is returned.

The LDAP\_SET\_TMPLITEM\_APPDATA() macro is used to set the value of the ti\_appdata field in a ldap\_tmplitem structure. This field is reserved for the calling application to use; it is not used internally.

The LDAP\_GET\_TMPLITEM\_APPDATA() macro is used to retrieve the value of the ti appdata field.

The LDAP\_IS\_TMPLITEM\_OPTION\_SET() macro is used to test a ldap\_tmplitem structure for the existence of an item option. The options currently defined are: LDAP\_DITEM\_OPT\_READONLY (this attribute should not be modified), LDAP\_DITEM\_OPT\_SORTVALUES (it makes sense to sort the values), LDAP\_DITEM\_OPT\_SINGLEVALUED (this attribute can only hold a single value), LDAP\_DITEM\_OPT\_VALUEREQUIRED (this attribute must contain at least one value), LDAP\_DITEM\_OPT\_HIDEIFEMPTY (do not show this item if there are no values), and LDAP\_DITEM\_OPT\_HIDEIFFALSE (for boolean attributes only: hide this item if the value is FALSE).

ldap\_next\_tmplcol() returns a pointer to the item (column) that follows column
col within row row of template tmpl.

DISPTMPL Structure Elements The ldap\_disptmpl structure is defined as:

```
struct
```

```
struct ldap disptmpl {
                              *dt_name;
   char
   char
                  *dt_pluralname;
   char
                             *dt iconname;
   unsigned long
                             dt_options;
   char
                             *dt authattrname;
   char
                             *dt defrdnattrname;
                             *dt defaddlocation;
   char
   struct ldap oclist *dt oclist;
   struct ldap_adddeflist *dt_adddeflist;
   struct ldap_tmplitem *dt_items;
   void *dt_appdata;
   struct ldap_disptmpl *dt_next;
};
```

The dt\_name member is the singular name of the template. The dt\_pluralname is the plural name. The dt\_iconname member will contain the name of an icon or other graphical element that can be used to depict entries that correspond to this display template. The dt\_options contains options which may be tested using the LDAP\_IS\_TMPLITEM\_OPTION\_SET() macro.

The dt\_authattrname contains the name of the DN-syntax attribute whose value(s) should be used to authenticate to make changes to an entry. If dt\_authattrname is NULL, then authenticating as the entry itself is appropriate. The dt\_defrdnattrname is the name of the attribute that is normally used to name entries of this type, for example, "cn" for person entries. The dt\_defaddlocation is the distinguished name of an entry below which new entries of this type are typically created (its value is site-dependent).

dt\_oclist is a pointer to a linked list of object class arrays, defined as:

```
struct ldap_oclist {
    char    **oc_objclasses;
    struct ldap_oclist    *oc_next;
};
```

These are used by the ldap\_oc2template() function.

dt\_adddeflist is a pointer to a linked list of rules for defaulting the values of attributes when new entries are created. The ldap\_adddeflist structure is defined as:

```
struct ldap_adddeflist {
    int        ad_source;
    char     *ad_attrname;
    char     *ad_value;
    struct ldap_adddeflist *ad_next;
};
```

The ad\_attrname member contains the name of the attribute whose value this rule sets. If ad\_source is LDAP\_ADSRC\_CONSTANTVALUE then the ad\_value member contains the (constant) value to use. If ad\_source is LDAP\_ADSRC\_ADDERSDN then ad\_value is ignored and the distinguished name of the person who is adding the new entry is used as the default value for ad\_attrname.

 TMPLITEM
 The ldap\_tmplitem structure is defined as:

 Structure Elements
 struct ldap\_tmplitem {

 unsigned long
 ti\_syntaxid;

 unsigned long
 ti\_options;

```
unsigned long ti_syntaxid;
unsigned long ti_options;
char *ti_attrname;
char *ti_label;
char **ti_args;
struct ldap_tmplitem *ti_next_in_row;
struct ldap_tmplitem *ti_next_in_col;
void *ti_appdata;
};
```

Syntax IDs

Syntax ids are found in the ldap\_tmplitem structure element ti\_syntaxid, and they can be used to determine how to display the values for the attribute associated

with an item. The LDAP\_GET\_SYN\_TYPE() macro can be used to return a general type from a syntax id. The five general types currently defined are: LDAP\_SYN\_TYPE\_TEXT (for attributes that are most appropriately shown as text), LDAP\_SYN\_TYPE\_IMAGE (for JPEG or FAX format images), LDAP\_SYN\_TYPE\_BOOLEAN (for boolean attributes), LDAP\_SYN\_TYPE\_BUTTON (for attributes whose values are to be retrieved and display only upon request, for example, in response to the press of a button, a JPEG image is retrieved, decoded, and displayed), and LDAP\_SYN\_TYPE\_ACTION (for special purpose actions such as "search for the entries where this entry is listed in the seeAlso attribute").

The LDAP\_GET\_SYN\_OPTIONS macro can be used to retrieve an unsigned long bitmap that defines options. The only currently defined option is LDAP\_SYN\_OPT\_DEFER, which (if set) implies that the values for the attribute should not be retrieved until requested.

There are sixteen distinct syntax ids currently defined. These generally correspond to one or more X.500 syntaxes.

LDAP\_SYN\_CASEIGNORESTR is used for text attributes which are simple strings whose case is ignored for comparison purposes.

LDAP\_SYN\_MULTILINESTR is used for text attributes which consist of multiple lines, for example, postalAddress, homePostalAddress, multilineDescription, or any attributes of syntax caseIgnoreList.

LDAP\_SYN\_RFC822ADDR is used for case ignore string attributes that are RFC-822 conformant mail addresses, for example, mail.

LDAP\_SYN\_DN is used for attributes with a Distinguished Name syntax, for example, seeAlso.

LDAP SYN BOOLEAN is used for attributes with a boolean syntax.

LDAP\_SYN\_JPEGIMAGE is used for attributes with a jpeg syntax, for example, jpegPhoto.

LDAP\_SYN\_JPEGBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve, decode, and display an attribute of jpeg syntax.

LDAP\_SYN\_FAXIMAGE is used for attributes with a photo syntax, for example, Photo. These are actually Group 3 Fax (T.4) format images.

LDAP\_SYN\_FAXBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve, decode, and display an attribute of photo syntax.

LDAP\_SYN\_AUDIOBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve and play an attribute of audio syntax. Audio values are in the "mu law" format, also known as "au" format.

LDAP\_SYN\_TIME is used for attributes with the UTCTime syntax, for example, lastModifiedTime. The value(s) should be displayed in complete date and time fashion.

LDAP\_SYN\_DATE is used for attributes with the UTCTime syntax, for example, lastModifiedTime. Only the date portion of the value(s) should be displayed.

LDAP SYN LABELEDURL is used for labeledURL attributes.

LDAP\_SYN\_SEARCHACTION is used to define a search that is used to retrieve related information. If ti\_attrname is not NULL, it is assumed to be a boolean attribute which will cause no search to be performed if its value is FALSE. The ti\_args structure member will have four strings in it: ti\_args [ 0 ] should be the name of an attribute whose values are used to help construct a search filter or "-dn" is the distinguished name of the entry being displayed should be used, ti\_args [ 1 ] should be a filter pattern where any occurrences of "%v" are replaced with the value derived from ti\_args [ 0 ], ti\_args [ 2 ] should be the name of an additional attribute to retrieve when performing the search, and ti\_args [ 3 ] should be a human-consumable name for that attribute. The ti\_args [ 2 ] attribute is typically displayed along with a list of distinguished names when multiple entries are returned by the search.

LDAP\_SYN\_LINKACTION is used to define a link to another template by name. ti\_args[0] will contain the name of the display template to use. The ldap\_name2template() function can be used to obtain a pointer to the correct ldap disptmpl structure.

LDAP\_SYN\_ADDDNACTION and LDAP\_SYN\_VERIFYDNACTION are reserved as actions but currently undefined.

**ERRORS** The init template functions return LDAP\_TMPL\_ERR\_VERSION if *buf* points to data that is newer than can be handled, LDAP\_TMPL\_ERR\_MEM if there is a memory allocation problem, LDAP\_TMPL\_ERR\_SYNTAX if there is a problem with the format of the templates buffer or file. LDAP\_TMPL\_ERR\_FILE is returned by ldap\_init\_templates if the file cannot be read. Other functions generally return NULL upon error.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Stability Level	Evolving

**SEE ALSO** | ldap(3LDAP), ldap\_entry2text(3LDAP), ldaptemplates.conf(4)

NAME	ldap_entry2text, ldap_entry2text_search, ldap_entry2html, ldap_entry2html_search, ldap_vals2html, ldap_vals2text – LDAP entry display functions		
SYNOPSIS	cc[ flag ] filelldap[ library ]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>int ldap_entry2text(LDAP *ld, char *buf, LDAPMessage *entry, struct ldap_disptmpl *tmpl, char **defattrs, char ***defvals, int (*writeproc)(), void *writeparm, char *eol, int rdncount, unsigned long opts);</pre>		
	<pre>int ldap_entry2text_search(LDAP *ld, char *dn, char *base, LDAPMessage *entry, struct ldap_disptmpl *tmpllist, char **defattrs, char ***defvals, int (*writeproc)(), void *writeparm, char *eol, int rdncount, unsigned long opts);</pre>		
	<pre>int ldap_vals2text(LDAP *ld, char *buf, char **vals, char *label, int     labelwidth, unsigned longsyntaxid, int (*writeproc)(), void *writeparm,     char *eol, int rdncount);</pre>		
	<pre>int ldap_entry2html(LDAP *ld, char *buf, LDAPMessage *entry, struct     ldap_disptmpl *tmpl, char **defattrs, char ***defvals, int     (*writeproc)(), void *writeparm, char *eol, int rdncount, unsigned     long opts, char *urlprefix, char *base);</pre>		
	<pre>int ldap_entry2html_search(LDAP *ld, char *dn, LDAPMessage *entry, struct ldap_disptmpl *tmpllist, char **defattrs, char ***defvals, int (*writeproc)(), void *writeparm, char *eol, int rdncount, unsigned long opts, char *urlprefix);</pre>		
	<pre>int ldap_vals2html(LDAP *ld, char *buf, char *vals, char *label, int</pre>		
	#define LDAP_DISP_OPT_AUTOLABELWIDTH 0x00000001		
	#define LDAP_DISP_OPT_HTMLBODYONLY 0x00000002		
	#define LDAP_DTMPL_BUFSIZ 2048		
DESCRIPTION	These functions use the LDAP display template functions (see <pre>ldap_disptmpl(3LDAP) and ldap_templates.conf(4)) to produce a plain text or an HyperText Markup Language (HTML) display of an entry or a set of values. Typical plain text output produced for an entry might look like:</pre>		
	"Barbara J Jensen, Information Technology Division" Also Known As: Babs Jensen Barbara Jensen Barbara J Jensen E-Mail Address: bjensen@terminator.rs.itd.umich.edu		

#### ldap\_entry2text(3LDAP)

```
Work Address:
535 W. William
Ann Arbor, MI 48103
Title:
Mythical Manager, Research Systems
...
```

The exact output produced will depend on the display template configuration. HTML output is similar to the plain text output, but more richly formatted.

ldap\_entry2text() produces a text representation of entry and writes the text by calling the writeproc function. All of the attributes values to be displayed must be present in entry; no interaction with the LDAP server will be performed within ldap\_entry2text. ld is the LDAP pointer obtained by a previous call to ldap\_open. writeproc should be declared as:

```
int writeproc( writeparm, p, len )
void *writeparm;
char *p;
int len;
```

where *p* is a pointer to text to be written and *len* is the length of the text. *p* is guaranteed to be zero-terminated. Lines of text are terminated with the string *eol. buf* is a pointer to a buffer of size LDAP DTMPL BUFSIZ or larger. If *buf is* NULL then a buffer is allocated and freed internally. *tmpl* is a pointer to the display template to be used (usually obtained by calling ldap oc2template). If *tmpl* is NULL, no template is used and a generic display is produced. *defattrs* is a NULL-terminated array of LDAP attribute names which you wish to provide default values for (only used if entry contains no values for the attribute). An array of NULL-terminated arrays of default values corresponding to the attributes should be passed in *defvals*. The rdncount parameter is used to limit the number of Distinguished Name (DN) components that are actually displayed for DN attributes. If *rdncount* is zero, all components are shown. opts is used to specify output options. The only values currently allowed are zero (default output), LDAP DISP OPT AUTOLABELWIDTH which causes the width for labels to be determined based on the longest label in tmpl, and LDAP DISP OPT HTMLBODYONLY. The LDAP DISP OPT HTMLBODYONLY option instructs the library not to include <HTML>, <HEAD>, <TITLE>, and <BODY> tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.

ldap\_entry2text\_search() is similar to ldap\_entry2text, and all of the like-named parameters have the same meaning except as noted below. If base is not NULL, it is the search base to use when executing search actions. If it is NULL, search action template items are ignored. If entry is not NULL, it should contain the objectClass attribute values for the entry to be displayed. If entry is NULL, dn must not be NULL, and ldap\_entry2text\_search will retrieve the objectClass values itself by calling ldap\_search\_s.ldap\_entry2text\_search will determine the appropriate display template to use by calling ldap\_oc2template, and will call ldap\_search\_s to retrieve any attribute values to be displayed. The tmpllist parameter is a pointer to the entire list of templates available (usually obtained by

	$\mathbf{I} = -$
	calling ldap_init_templates or ldap_init_templates_buf). If <i>tmpllist</i> is NULL, ldap_entry2text_search will attempt to read a load templates from the default template configuration file ETCDIR/ldaptemplates.conf.
	<pre>ldap_vals2text produces a text representation of a single set of LDAP attribute values. The <i>ld</i>, <i>buf</i>, <i>writeproc</i>, <i>writeparm</i>, <i>eol</i>, and <i>rdncount</i> parameters are the same as the like-named parameters for ldap_entry2text. <i>vals</i> is a NULL-terminated list of values, usually obtained by a call to ldap_get_values. <i>label</i> is a string shown next to the values (usually a friendly form of an LDAP attribute name). <i>labelwidth</i> specifies the label margin, which is the number of blank spaces displayed to the left of the values. If zero is passed, a default label width is used. <i>syntaxid</i> is a display template attribute syntax identifier (see ldap_disptmpl(3LDAP) for a list of the pre-defined LDAP_SYN values).</pre>
	<pre>ldap_entry2html produces an HTML representation of entry. It behaves exactly like ldap_entry2text(3LDAP), except for the formatted output and the addition of two parameters. urlprefix is the starting text to use when constructing an LDAP URL. The default is the string ldap:/// The second additional parameter, base, the search base to use when executing search actions. If it is NULL, search action template items are ignored.</pre>
	<pre>ldap_entry2html_search behaves exactly like ldap_entry2text_search(3LDAP), except HTML output is produced and one additional parameter is required. <i>urlprefix</i> is the starting text to use when constructing an LDAP URL. The default is the string <i>ldap:///</i></pre>
	<pre>ldap_vals2html behaves exactly like ldap_vals2text,exceptHTMLoutputis and one additional parameter is required. urlprefix is the starting text to use when constructing an LDAP URL. The default is the string ldap:///</pre>
ERRORS	These functions all return an LDAP error code ( LDAP_SUCCESS is returned if no error occurs). See ldap_error(3LDAP) for details. The <i>ld_errno</i> field of the ld parameter is also set to indicate the error.
FILES	ETCDIR/ldaptemplates.conf
ATTRIBUTES	See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** ldap(3LDAP), ldap\_disptmpl(3LDAP), ldaptemplates.conf(4)

ldap_	_error(3LDAP)
-------	---------------

SYNOPSIScc[flag] file] -11dap[library]#include <lber.h> #include <ldap.h>struct ldap_error(int e_code, char *e_reason); struct ldap_error(int e_code, char *e_reason); struct ldap_error(LDAP *ld, char *s); int ldap_perror(LDAP *ld, char *s); int ldap_result2error(LDAP *ld, LDAPMessage *res, int freeit);DESCRIPTIONDESCRIPTIONThese functions provide interpretation of the various error codes returned by the LDAP portocol and LDAP library functions and assigned to an error field in the ld structure. ldap_perror() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead.The ldap_result2error() function takes res, a result as produced by ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error code are listed below. If the freeit parameter is no zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in 1d is set and returned.The returned value can be passed to ldap_err2string() or looked up in ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_err1ist[] array is signaled by an error code of -1. The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.ERRORSThe possible values for an ldap error code are: LDAD GUERDE</ldap.h></lber.h>	NAME	ldap_error, ldap_perror, ldap_result2error protocol error handling functions	r, ldap_errlist, ldap_err2string – LDAP
<pre>#include <ldap.h> struct ldap_error(int e_code, char *e_reason); struct ldap_error(ldap_errlist[]; char *ldap_err2string(int err); void ldap_perror(LDAP *ld, char *s); int ldap_result2error(LDAP *ld, LDAPMessage *res, int freeit);  DESCRIPTION These functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to an error field in the ld structure.ldap_error() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead. The ldap_result2error() function takes res, a result as produced by ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error code as are listed below. If the freeit parameter is non zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in 1d is set and returned. The returned value can be passed to ldap_err2string() or looked up in ldap_errlist[] to get a text description of the message. The string returned from ldap_errlist[] to get new a static area that should not be modified. The last element in the ldap_errlist[] array is signaled by an error code of -1. The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works. ERRORS The possible values for an ldap error code are:</ldap.h></pre>	SYNOPSIS	cc[ flag ] filelldap[ library ]	
FRRORSstruct ldap_error() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.			
<pre>char *ldap_err2string(int err); void ldap_perror(LDAP *ld, char *s); int ldap_result2error(LDAP *ld, LDAPMessage *res, int freeit);</pre> DESCRIPTION These functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to an error field in the ld structure. ldap_perror() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead. The ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the freeit parameter is non zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in ld is set and returned. The returned value can be passed to ldap_err2string() or looked up in ldap_errlist[] to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_errlist[] array is signaled by an error code of -1. The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works. ERRORS The possible values for an ldap error code are:		<pre>struct ldap_error(int e_code, cha</pre>	ar *e_reason);
<ul> <li>void ldap_perror (LDAP *ld, char *s); int ldap_result2error (LDAP *ld, LDAPMessage *res, int freeit);</li> <li>DESCRIPTION</li> <li>These functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to an error field in the ld structure. ldap_perror() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead.</li> <li>The ldap_result2error() function takes res, a result as produced by ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the freeit parameter is non zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in ld is set and returned.</li> <li>The returned value can be passed to ldap_err2string() or looked up in ldap_err1ist[] to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_err1ist[] array is signaled by an error code of -1.</li> <li>The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.</li> <li>ERRORS</li> <li>The possible values for an ldap error code are:</li> </ul>		<pre>struct ldaperror ldap_errlist[];</pre>	
DESCRIPTIONInt ldap_result2error(LDAP *ld, LDAPMessage *res, int freeit);DESCRIPTIONThese functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to an error field in the ld structure. ldap_perror() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead.The ldap_result2error() function takes res, a result as produced by ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the freeit parameter is non zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in ld is set and returned.The returned value can be passed to ldap_err2string() or looked up in ldap_err1ist[] to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_errlist[] array is signaled by an error code of -1.The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.ERRORSThe possible values for an ldap error code are:		<pre>char *ldap_err2string(int err);</pre>	
DESCRIPTIONThese functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to an error field in the 1d structure. 1dap_perror() and 1dap_result2error() functions are deprecated for all new development; 1dap_err2string() should be used instead.The 1dap_result2error() function takes <i>res</i> , a result as produced by 1dap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the <i>freeit</i> parameter is non zero it indicates that the <i>res</i> parameter should be freed by a call to 1dap_msgfree(3LDAP) after the error code has been extracted. The error field in 1d is set and returned.The returned value can be passed to 1dap_err2string() or looked up in 1dap_err1ist[] to get a text description of the message. The string returned from 1dap_err2string() is a pointer to a static area that should not be modified. The last element in the 1dap_err1ist[] array is signaled by an error code of -1.The 1dap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.ERRORSThe possible values for an 1dap error code are:		void <b>ldap_perror</b> (LDAP * <i>ld</i> , char *	*s);
LDAP protocol and LDAP library functions and assigned to an error field in the ld structure. ldap_perror() and ldap_result2error() functions are deprecated for all new development; ldap_err2string() should be used instead.The ldap_result2error() function takes res, a result as produced by ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the <i>freeit</i> parameter is non zero it indicates that the res parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in ld is set and returned.The returned value can be passed to ldap_err2string() or looked up in ldap_errlist[] to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_errlist[] array is signaled by an error code of -1.The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.ERRORSThe possible values for an ldap error code are:		int <b>ldap_result2error</b> (LDAP * <i>ld</i> ,	LDAPMessage *res, int freeit);
<pre>ldap_result(3LDAP) or other synchronous LDAP calls, and returns the corresponding error code. Possible error codes are listed below. If the <i>freeit</i> parameter is non zero it indicates that the <i>res</i> parameter should be freed by a call to ldap_msgfree(3LDAP) after the error code has been extracted. The error field in 1d is set and returned. The returned value can be passed to ldap_err2string() or looked up in ldap_errlist[] to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the ldap_errlist[] array is signaled by an error code of -1. The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.</pre>	DESCRIPTION	LDAP protocol and LDAP library function structure. ldap_perror() and ldap_reference.	ns and assigned to an error field in the ld esult2error() functions are deprecated
ldap_errlist[] to get a text description of the message. The string returned fromldap_err2string() is a pointer to a static area that should not be modified. Thelast element in the ldap_errlist[] array is signaled by an error code of -1.The ldap_perror() function can be called to print an indication of the error onstandard error, similar to the way perror(3C) works.ERRORSThe possible values for an ldap error code are:		<pre>ldap_result(3LDAP) or other synchron corresponding error code. Possible error of is non zero it indicates that the res paramo ldap_msgfree(3LDAP) after the error compared to the error of the error of the erro</pre>	nous LDAP calls, and returns the codes are listed below. If the <i>freeit</i> parameter eter should be freed by a call to
standard error, similar to the way perror(3C) works.ERRORSThe possible values for an Idap error code are:		<pre>ldap_errlist[ ] to get a text descript ldap_err2string() is a pointer to a st</pre>	ion of the message. The string returned from attic area that should not be modified. The
		—	-
	ERRORS	The possible values for an ldap error code	e are:
LDAP_SUCCESS Ine request was successful.		LDAP_SUCCESS	The request was successful.
LDAP_OPERATIONS_ERROR An operations error occurred.		LDAP_OPERATIONS_ERROR	An operations error occurred.
IDAD DROTOCOL EPROP A protocol violation was detected		LDAP_PROTOCOL_ERROR	A protocol violation was detected.
		LDAP_TIMELIMIT_EXCEEDED	An LDAP time limit was exceeded.
		LDAP_SIZELIMIT_EXCEEDED	An LDAP size limit was exceeded.
LDAP_TIMELIMIT_EXCEEDED An LDAP time limit was exceeded.		LDAP_COMPARE_FALSE	A compare operation returned false.
LDAP_TIMELIMIT_EXCEEDEDAn LDAP time limit was exceeded.LDAP_SIZELIMIT_EXCEEDEDAn LDAP size limit was exceeded.LDAP_COMPARE_FALSEA compare operation returned false.		LDAP_COMPARE_TRUE	A compare operation returned true.
LDAP_OPERATIONS_ERROR An operations error occurred.	ERRORS	last element in the ldap_errlist[] at The ldap_perror() function can be cal standard error, similar to the way perror The possible values for an ldap error code LDAP_SUCCESS LDAP_OPERATIONS_ERROR LDAP_PROTOCOL_ERROR LDAP_TIMELIMIT_EXCEEDED	rray is signaled by an error code of -1. led to print an indication of the error on r(3C) works. e are: The request was successful. An operations error occurred. A protocol violation was detected. An LDAP time limit was exceeded.
I.D.A.D. DDOTOCOL. EDDOD A protocol violation was detected			-
		LDAP_SIZELIMIT_EXCEEDED	An LDAP size limit was exceeded.
LDAP_TIMELIMIT_EXCEEDED An LDAP time limit was exceeded.		LDAP_COMPARE_FALSE	A compare operation returned false.
LDAP_TIMELIMIT_EXCEEDEDAn LDAP time limit was exceeded.LDAP_SIZELIMIT_EXCEEDEDAn LDAP size limit was exceeded.		LDAP_COMPARE_TRUE	A compare operation returned true.
LDAP_TIMELIMIT_EXCEEDEDAn LDAP time limit was exceeded.LDAP_SIZELIMIT_EXCEEDEDAn LDAP size limit was exceeded.LDAP_COMPARE_FALSEA compare operation returned false.			

LDAP_STRONG_AUTH_NOT_SUPPORTED	The LDAP server does not support strong authentication.
LDAP_STRONG_AUTH_REQUIRED	Strong authentication is required for the operation.
LDAP_PARTIAL_RESULTS	Partial results only returned.
LDAP_NO_SUCH_ATTRIBUTE	The attribute type specified does not exist in the entry.
LDAP_UNDEFINED_TYPE	The attribute type specified is invalid.
LDAP_INAPPROPRIATE_MATCHING	Filter type not supported for the specified attribute.
LDAP_CONSTRAINT_VIOLATION	An attribute value specified violates some constraint (for example, a postalAddress has too many lines, or a line that is too long).
LDAP_TYPE_OR_VALUE_EXISTS	An attribute type or attribute value specified already exists in the entry.
LDAP_INVALID_SYNTAX	An invalid attribute value was specified.
LDAP_NO_SUCH_OBJECT	The specified object does not exist in The Directory.
LDAP_ALIAS_PROBLEM	An alias in The Directory points to a nonexistent entry.
LDAP_INVALID_DN_SYNTAX	A syntactically invalid DN was specified.
LDAP_IS_LEAF	The object specified is a leaf.
LDAP_ALIAS_DEREF_PROBLEM	A problem was encountered when dereferencing an alias.
LDAP_INAPPROPRIATE_AUTH	Inappropriate authentication was specified (for example, LDAP_AUTH_SIMPLE was specified and the entry does not have a userPassword attribute).
LDAP_INVALID_CREDENTIALS	Invalid credentials were presented (for example, the wrong password).
LDAP_INSUFFICIENT_ACCESS	The user has insufficient access to perform the operation.
LDAP_BUSY	The DSA is busy.
LDAP_UNAVAILABLE	The DSA is unavailable.
LDAP_UNWILLING_TO_PERFORM	The DSA is unwilling to perform the operation.

## ldap\_error(3LDAP)

p_error(3LDAP)		
	LDAP_LOOP_DETECT	A loop was detected.
	LDAP_NAMING_VIOLATION	A naming violation occurred.
	LDAP_OBJECT_CLASS_VIOLATION	An object class violation occurred (for example, a "must" attribute was missing from the entry).
	LDAP_NOT_ALLOWED_ON_NONLEAF	The operation is not allowed on a nonleaf object.
	LDAP_NOT_ALLOWED_ON_RDN	The operation is not allowed on an RDN.
	LDAP_ALREADY_EXISTS	The entry already exists.
	LDAP_NO_OBJECT_CLASS_MODS	Object class modifications are not allowed.
	LDAP_OTHER	An unknown error occurred.
	LDAP_SERVER_DOWN	The LDAP library can't contact the LDAP server.
	LDAP_LOCAL_ERROR	Some local error occurred. This is usually a failed malloc.
	LDAP_ENCODING_ERROR	An error was encountered encoding parameters to send to the LDAP server.
	LDAP_DECODING_ERROR	An error was encountered decoding a result from the LDAP server.
	LDAP_TIMEOUT	A timelimit was exceeded while waiting for a result.
	LDAP_AUTH_UNKNOWN	The authentication method specified to ldap_bind() is not known.
	LDAP_FILTER_ERROR	An invalid filter was supplied to ldap_search( ) (for example, unbalanced parentheses).
	LDAP_PARAM_ERROR	An ldap function was called with a bad parameter (for example, a NULL ld pointer, etc.).
	LDAP_NO_MEMORY	An memory allocation (for example, malloc(3N)) call failed in an ldap library function.
ATTRIBUTES	See attributes(5) for a description of the	ne following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE

SUNWlldap (32-bit)

320 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

Availability

# ldap\_error(3LDAP)

	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** attributes(5), ldap(3LDAP), perror(3C)

# ldap\_first\_attribute(3LDAP)

NAME	ldap_first_attribute, ldap_next_attribute - s	step through LDAP entry attributes
SYNOPSIS	cc[ flag ] filelldap[ library ]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>char *ldap_first_attribute(LDAP</pre>	*ld, LDAPMessage * <i>entry</i> , BerElement
	<pre>char *ldap_next_attribute(LDAP *l</pre>	d, LDAPMessage * <i>entry</i> , BerElement
DESCRIPTION	<b>TION</b> The ldap_first_attribute() and ldap_next_attribute() functions are use to step through the attributes in an LDAP entry. ldap_first_attribute() takes an <i>entry</i> as returned by ldap_first_entry(3LDAP) or ldap_next_entry(3LDA and returns a pointer to a per-connection buffer containing the first attribute type in the entry. The return value should be treated as if it is a pointer to a static area (that i strdup(3C) it if you want to save it).	
	current position. This pointer should be pa ldap_next_attribute() and is used us attributes. This pointer is freed by ldap_next_ more attributes (that is, when ldap_next_ the caller is responsible for freeing the Ber	<pre>sed to effectively step through the entry's ext_attribute() when there are no _attribute() returns NULL). Otherwise, Element pointed to by <i>berptr</i> when it is no P). When calling ber_free(3LDAP) in this</pre>
	The attribute names returned are suitable f ldap_get_values(3LDAP) to retrieve the	
ERRORS	If an error occurs, NULL is returned and the to indicate the error. See ldap_error(3LD codes.	e ld_errno field in the ld parameter is set DAP) for a description of possible error
ATTRIBUTES	See attributes(5) for a description of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWIIdap (32-bit)
		SUNWldapx (64-bit)
	Stability Level	Evolving
SEE ALSO	ldap(3LDAP),ldap_first_entry(3LDA ldap_error(3LDAP)	AP),ldap_get_values(3LDAP),

ldap\_first\_attribute(3LDAP)

**NOTES** | The ldap\_first\_attribute() function mallocs memory that may need to be freed by the caller via ber\_free(3LDAP).

ldap\_first\_entry(3LDAP)

NAME	ldap_first_entry, ldap_next_entry, ldap_count_entries, ldap_count_references, ldap_first_reference, ldap_next_reference – LDAP entry parsing and counting functions
SYNOPSIS	cc[ flag ] filelldap[ library ]
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
	LDAPMessage <b>*ldap_first_entry</b> (LDAP* <i>ld</i> , LDAPMessage * <i>result</i> );
	LDAPMessage <b>*ldap_next_entry</b> (LDAP * $ld$ , LDAPMessage * $entry$ );
	<pre>ldap_count_entries(LDAP *ld, LDAPMessage *result);</pre>
	LDAPMessage <b>*ldap_first_reference</b> (LDAP *ld, LDAPMessage *res);
	LDAPMessage <b>*ldap_next_reference</b> (LDAP <b>*</b> <i>ld</i> , LDAPMessage <b>*</b> <i>res</i> );
	int <b>ldap_count_references</b> (LDAP * <i>ld</i> , LDAPMessage * <i>res</i> );
DESCRIPTION	These functions are used to parse results received from ldap_result(3LDAP) or the synchronous LDAP search operation functions ldap_search_s(3LDAP) and ldap_search_st(3LDAP).
	The ldap_first_entry() function is used to retrieve the first entry in a chain of search results. It takes the <i>result</i> as returned by a call to ldap_result(3LDAP) or ldap_search_s(3LDAP) or ldap_search_st(3LDAP) and returns a pointer to the first entry in the result.
	This pointer should be supplied on a subsequent call to <pre>ldap_next_entry()</pre> to get the next entry, the result of which should be supplied to the next call to <pre>ldap_next_entry()</pre> , etc. <pre>ldap_next_entry()</pre> will return NULL when there are no more entries. The entries returned from these calls are used in calls to the functions described in <pre>ldap_get_dn(3LDAP)</pre> , <pre>ldap_first_attribute(3LDAP)</pre> , <pre>ldap_get_values(3LDAP)</pre> , etc.
	A count of the number of entries in the search result can be obtained by calling ldap_count_entries().
	<pre>ldap_first_reference() and ldap_next_reference() are used to step through and retrieve the list of continuation references from a search result chain.</pre>
	The ldap_count_references() function is used to count the number of references that are contained in and remain in a search result chain.
ERRORS	If an error occurs in ldap_first_entry() or ldap_next_entry(), NULL is returned and the ld_errno field in the ld parameter is set to indicate the error. If an error occurs in ldap_count_entries(), -1 is returned, and ld_errno is set appropriately. See ldap_error(3LDAP) for a description of possible error codes.
ATTRIBUTES	See attributes(5) for a description of the following attributes:

ldap\_first\_entry(3LDAP)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

## SEE ALSO ldap(3LDAP), ldap\_result(3LDAP), ldap\_search(3LDAP), ldap\_first\_attribute(3LDAP), ldap\_get\_values(3LDAP), ldap\_get\_dn(3LDAP)

ldap\_first\_message(3LDAP)

NAME	ldap_first_message, ldap_count_messages, ldap_next_message, ldap_msgtype – LDAP message processing functions		
SYNOPSIS	cc[flag] filelldap[library]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	int ldap_count_messages(LDAP $*ld$ ,	LDAPMessage *res);	
	LDAPMessage <b>*ldap_first_message</b> (	LDAP $^{*ld}$ , LDAPMessage $^{*res}$ );	
	LDAPMessage <b>*ldap_next_message</b> (L	DAP $*ld$ , LDAPMessage $*msg$ );	
	int <b>ldap_msgtype</b> (LDAPMessage *res)	;	
DESCRIPTION	<pre>ldap_count_messages() is used to counchain of results if called with a message, en ldap_first_message(), ldap_next_m ldap_next_entry(), ldap_first_ref ldap_next_reference()</pre>	<pre>try, or reference returned by essage(),ldap_first_entry(),</pre>	
	<pre>ldap_first_message() and ldap_nex through the list of messages in a result chai</pre>		
	ldap_msgtype() function returns the typ	e of an LDAP message.	
<b>RETURN VALUES</b>	<pre>ldap_first_message() and ldap_nex can include referral messages, entry messages</pre>	t_message() return LDAPMessage which ges and result messages.	
	ldap_count_messages() returns the nu results.	mber of messages contained in a chain of	
ERRORS	<pre>ldap_first_message() and ldap_next_message() return NULL when no more messages exist. NULL is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle ld will be set to indicate the error.</pre>		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	
	Stability Level	Evolving	
SEE ALSO	ldap_error(3LDAP),ldap_result(3LD	OAP), attributes(5)	

NAME	ldap_friendly, ldap_friendly_name, ldap_free_friendlymap – LDAP attribute remapping functions		
SYNOPSIS	<pre>cc[flag] filelldap[ library] #include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>char *ldap_friendly_name(char *file</pre>	<i>ename</i> , char * <i>name</i> , FriendlyMap	
	void <pre>ldap_free_friendlymap</pre> (Frien	dlyMap **map);	
DESCRIPTION	This function is used to map one set of strings to another. Typically, this is done for country names, to map from the two-letter country codes to longer more readable names. The mechanism is general enough to be used with other things, though.		
	<i>filename</i> is the name of a file containing the unfriendly to friendly mapping, <i>name</i> is the unfriendly name to map to a friendly name, and <i>map</i> is a result-parameter that should be set to NULL on the first call. It is then used to hold the mapping in core so that the file need not be read on subsequent calls.		
	For example:		
	<pre>FriendlyMap *map = NULL; printf( "unfriendly %s =&gt; friendly %s\n", name,</pre>		
	The mapping file should contain lines like this: unfriendlyname\tfriendlyname. Lines that begin with a '#' character are comments and are ignored.		
	The ldap_free_friendlymap() call is used to free structures allocated by ldap_friendly_name() when no more calls to ldap_friendly_name() are to be made.		
ERRORS	NULL is returned by ldap_friendly_name() if there is an error opening <i>filename</i> , or if the file has a bad format, or if the <i>map</i> parameter is NULL.		
FILES	ETCDIR/ldapfriendly.conf		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	

Evolving

SEE ALSO | ldap(3LDAP)

Stability Level

Networking Library Functions 327

# ldap\_get\_dn(3LDAP)

NAME	ldap_get_dn, ldap_explode_dn, ldap_dn2ufn, ldap_is_dns_dn, ldap_explode_dns, ldap_dns_to_dn – LDAP DN handling functions
SYNOPSIS	cc[flag] filelldap[ library]
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
	char <b>*ldap_get_dn</b> (LDAP <i>*ld</i> , LDAPMessage <i>*entry</i> );
	char <b>**ldap_explode_dn</b> (char <b>*</b> <i>dn</i> , int <i>notypes</i> );
	<pre>char *ldap_dn2ufn(char *dn);</pre>
	<pre>int ldap_is_dns_dn(char *dn);</pre>
	<pre>char **ldap_explode_dns(char *dn);</pre>
	<pre>char *ldap_dns_to_dn(char *dns_name, int *nameparts);</pre>
DESCRIPTION	These functions allow LDAP entry names (Distinguished Names, or DNs) to be obtained, parsed, converted to a user-friendly form, and tested. A DN has the form described in RFC 1779 <i>A String Representation of Distinguished Names</i> , unless it is an experimental DNS-style DN which takes the form of an RFC 822 mail address.
	The ldap_get_dn() function takes an <i>entry</i> as returned by ldap_first_entry(3LDAP) or ldap_next_entry(3LDAP) and returns a copy of the entry's DN. Space for the DN will have been obtained via malloc(3C), and should be freed by the caller by a call to free(3C).
	The ldap_explode_dn() function takes a DN as returned by ldap_get_dn() and breaks it up into its component parts. Each part is known as a Relative Distinguished Name, or RDN. ldap_explode_dn() returns a NULL-terminated array, each component of which contains an RDN from the DN. The <i>notypes</i> parameter is used to request that only the RDN values be returned, not their types. For example, the DN "cn=Bob, c=US" would return as either { "cn=Bob", "c=US", NULL } or { "Bob", "US", NULL }, depending on whether notypes was 0 or 1, respectively. The result can be freed by calling ldap_value_free(3LDAP).
	<pre>ldap_dn2ufn() is used to turn a DN as returned by ldap_get_dn() into a more user-friendly form, stripping off type names. See RFC 1781 "Using the Directory to Achieve User Friendly Naming" for more details on the UFN format. The space for the UFN returned is obtained by a call to malloc(3C), and the user is responsible for freeing it via a call to free(3C).</pre>
	ldap_is_dns_dn() returns non-zero if the dn string is an experimental DNS-style DN (generally in the form of an RFC 822 e-mail address). It returns zero if the dn appears to be an RFC 1779 format DN.
	ldap_explode_dns() takes a DNS-style DN and breaks it up into its component parts. ldap_explode_dns() returns a NULL-terminated array. For example, the DN

"mcs.umich.edu" will return { "mcs", "umich", "edu", NULL }. The result can be freed by calling ldap\_value\_free(3LDAP).

ldap\_dns\_to\_dn() converts a DNS domain name into an X.500 distinguished name. A string distinguished name and the number of nameparts is returned.

**ERRORS** If an error occurs in ldap\_get\_dn(), NULL is returned and the ld\_errno field in the ld parameter is set to indicate the error. See ldap\_error(3LDAP) for a description of possible error codes. ldap\_explode\_dn(), ldap\_explode\_dns() and ldap\_dn2ufn() will return NULL with errno(3C) set appropriately in case of trouble.

If an error in ldap\_dns\_to\_dn() is encountered zero is returned. The caller should free the returned string if it is non-zero.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIldap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

- SEE ALSO | ldap(3LDAP), ldap\_first\_entry(3LDAP), ldap\_error(3LDAP), ldap\_value\_free(3LDAP)
  - **NOTES** These functions allocate memory that the caller must free.

# ldap\_get\_entry\_controls(3LDAP)

NAME	ldap_get_entry_controls – get the LDAP controls included with a directory entry in a set of search results		
SYNOPSIS	cc -flag <i>file</i> lldap [ -library ] #include <ldap.h></ldap.h>		
	int <b>ldap_get_entry_com</b> LDAPControl ***ser		*ld, LDAPMessage * <i>entry</i> ,
DESCRIPTION	The ldap_get_entry_controls() function retrieves the LDAP v3 controls included in a directory entry in a chain of search results. The LDAP controls are specified in an array of LDAPControl structures. Each LDAPControl structure represents an LDAP control. The function takes <i>entry</i> as a parameter, which points to an LDAPMessage structure that represents an entry in a chain of search results.		
	The entry notification controls that are used with persistent search controls are the only controls that are returned with individual entries. Other controls are returned with results sent from the server. You can call ldap_parse_result() to retrieve those controls.		
ERRORS	ldap_get_entry_controls() returns the following error codes.		
	LDAP_SUCCESS	LDAP control	ls were successfully retrieved.
	LDAP_DECODING_ERROR	An error occu message.	rred when decoding the BER-encoded
	LDAP_PARAM_ERROR	An invalid pa	arameter was passed to the function.
	LDAP_NO_MEMORY	Memory canr	not be allocated.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	Availability		SUNWcsl (32-bit)
			SUNWcslx (64-bit)

**SEE ALSO** | ldap\_error(3LDAP), ldap\_parse\_result(3LDAP), attributes(5)

Evolving

Interface Stability

NAME	ldap_getfilter, ldap_init_getfilter, ldap_init_getfilter_buf, ldap_getfilter_free, ldap_getfirstfilter, ldap_getnextfilter, ldap_setfilteraffixes, ldap_build_filter – LDAP filter generating functions
SYNOPSIS	cc[ flag ] filelldap[ library ]
	<pre>#include <lber.h> #include <ldap.h> #define LDAP_FILT_MAXSIZ 1024</ldap.h></lber.h></pre>
	LDAPFiltDesc *ldap_init_getfilter(char *file);
	LDAPFiltDesc *ldap_init_getfilter_buf(char *buf, long buflen);
	<pre>ldap_getfilter_free(LDAPFiltDesc *lfdp);</pre>
	LDAPFiltInfo <b>*ldap_getfirstfilter</b> (LDAPFiltDesc <i>*lfdp</i> , char <i>*tagpat</i> , char <i>*value</i> );
	LDAPFiltInfo *ldap_getnextfilter(LDAPFiltDesc *lfdp);
	<pre>void ldap_setfilteraffixes(LDAPFiltDesc *lfdp, char *prefix, char *suffix);</pre>
	<pre>void ldap_build_filter(char *buf, unsigned long buflen, char *pattern, char *prefix, char *suffix, char *attr, char *value, char **valwords);</pre>
DESCRIPTION	These functions are used to generate filters to be used in ldap_search(3LDAP) or ldap_search_s(3LDAP). Either ldap_init_getfilter or ldap_init_getfilter_buf must be called prior to calling any of the other functions except ldap_build_filter.
	<pre>ldap_init_getfilter() takes a file name as its only argument. The contents of the file must be a valid LDAP filter configuration file (see ldapfilter.conf(4)). If the file is successfully read, a pointer to an LDAPFiltDesc is returned. This is an opaque object that is passed in subsequent get filter calls.</pre>
	<pre>ldap_init_getfilter_buf() reads from buf (whose length is buflen) the LDAP filter configuration information. buf must point to the contents of a valid LDAP filter configuration file (see ldapfilter.conf(4)). If the filter configuration information is successfully read, a pointer to an LDAPFiltDesc is returned. This is an opaque object that is passed in subsequent get filter calls.</pre>
	<pre>ldap_getfilter_free() deallocates the memory consumed by ldap_init_getfilter. Once it is called, the LDAPFiltDesc is no longer valid and cannot be used again.</pre>
	<pre>ldap_getfirstfilter() retrieves the first filter that is appropriate for value. Only filter sets that have tags that match the regular expession tagpat are considered. ldap_getfirstfilter returns a pointer to an LDAPFiltInfo structure, which contains a filter with value inserted as appropriate in lfi_filter, a text match description in lfi_desc, lfi_scope set to indicate the search scope, and</pre>

#### ldap\_getfilter(3LDAP)

lfi\_isexact set to indicate the type of filter. NULL is returned if no matching filters
are found.lfi\_scope will be one of LDAP\_SCOPE\_BASE, LDAP\_SCOPE\_ONELEVEL,
or LDAP\_SCOPE\_SUBTREE.lfi\_isexact will be zero if the filter has any '~' or '\*'
characters in it and non-zero otherwise.

ldap\_getnextfilter() retrieves the next appropriate filter in the filter set that was
determined when ldap\_getfirstfilter was called. It returns NULL when the list
has been exhausted.

ldap\_setfilteraffixes() sets a prefix to be prepended and a suffix to be
appended to all filters returned in the future.

ldap\_build\_filter() constructs an LDAP search filter in buf. buflen is the size, in bytes, of the largest filter buf can hold. A pattern for the desired filter is passed in pattern. Where the string %a appears in the pattern it is replaced with attr. prefix is pre-pended to the resulting filter, and suffix is appended. Either can be NULL (in which case they are not used). value and valwords are used when the string %v appears in pattern. See ldapfilter.conf(4) for a description of how %v is handled.

- **ERRORS** NULL is returned by ldap\_init\_getfilter if there is an error reading *file*. NULL is returned by ldap\_getfirstfilter and ldap\_getnextfilter when there are no more appropriate filters to return.
  - FILES ETCDIR/ldapfilter.conf LDAP filtering routine configuration file.
- **ATTRIBUTES** See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

- **SEE ALSO** | ldap(3LDAP), ldapfilter.conf(4)
  - **NOTES** The return values for all of these functions are declared in the <ldap.h> header file. Some functions may allocate memory which must be freed by the calling application.

NAME	ldap_get_lang_values, ldap_get_lang_values_len – return an attribute's values that
INAMIL	matches a specified language subtype
SYNOPSIS	cc -flag <i>file</i> lldap [ -library ] #include <ldap.h></ldap.h>
	<pre>char **ldap_get_lang_values(LDAP *ld, LDAPMessage *entry, const char *target, char **type);</pre>
	<pre>struct berval **ldap_get_lang_values_len(LDAP *ld, LDAPMessage *entry, const char *target, char **type);</pre>
DESCRIPTION	The ldap_get_lang_values() function returns an array of an attribute's string values that matches a specified language subtype. To retrieve the binary data from an attribute, call the ldap_get_lang_values_len() function instead.
	<pre>ldap_get_lang_values() should be called to retrieve a null-terminated array of an attribute's string values that match a specified language subtype. The <i>entry</i> parameter is the entry retrieved from the directory. The <i>target</i> parameter should contain the attribute type the values that are required, including the optional language subtype. The <i>type</i> parameter points to a buffer that returns the attribute type retrieved by this function. Unlike the ldap_get_values() function, if a language subtype is specified, this function first attempts to find and return values that match that subtype, for example, cn;lang-en.</pre>
	<pre>ldap_get_lang_values_len() returns a null-terminated array of pointers to berval structures, each containing the length and pointer to a binary value of an attribute for a given entry. The <i>entry</i> parameter is the result returned by ldap_result() or ldap_search_s() functions. The <i>target</i> parameter is the attribute returned by the call to ldap_first_attribute() or ldap_next_attribute(), or the attribute as a literal string, such as jpegPhoto or audio.</pre>
	These functions are deprecated. Use ldap_get_values() or ldap_get_values_len() instead.
RETURN VALUES	If successful, ldap_get_lang_values() returns a null-terminated array of the attribute's values. If the call is unsuccessful, or if no such attribute exists in the <i>entry</i> , it returns a NULL and sets the appropriate error code in the LDAP structure.
	The ldap_get_lang_values_len() function returns a null-terminated array of pointers to berval structures, which in turn, if successful, contain pointers to the attribute's binary values. If the call is unsuccessful, or if no such attribute exists in the <i>entry</i> , it returns a NULL and sets the appropriate error code in the LDAP structure.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ldap\_get\_lang\_values(3LDAP)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Obsolete

SEE ALSO ldap\_first\_attribute(3LDAP), ldap\_next\_attribute(3LDAP), ldap\_get\_values(3LDAP), ldap\_result(3LDAP), ldap\_search\_s(3LDAP), attributes(5)

334 man pages section 3: Networking Library Functions • Last Revised 25 Oct 2001

NAME	ldap_get_option, ldap_set_option – get/set session preferences in the ldap structure.		
SYNOPSIS	cc[ flag ] filelldap[ library ]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	LDAP <b>ldap_set_option</b> (LDA	<pre>P *ld, int option, void *optdata[]);</pre>	
	LDAP <b>ldap_get_option</b> (LDA	<pre>P *ld, int option, void optdata[]);</pre>	
DESCRIPTION	ldap_get_option() gets sess	to session preferences to an LDAP structure. ion preferences from the LDAP structure. ion preferences in the LDAP structure.	
	information about the connectio	ch is a pointer to an LDAP structure containing n to the LDAP server. <i>option</i> is the name of the option s a pointer to the value of the option that you want to	
	The option parameter can have o	ne of the values listed in the following section.	
PARAMETERS	The following are the values for the <i>option</i> parameter:		
	LDAP_OPT_API_INFO Used to retrieve some basic information about the LDAP API implementation at execution time. The data type for the <i>optdata</i> parameter is (LDAPAPIInfo *). This option is READ-ONLY and cannot be set.		
	LDAP_OPT_DEREF Determines how aliases are handled during a search . The data type for the <i>optdata</i> parameter is (int *). <i>optdata</i> can be one of the following values:		
	LDAP_DEREF_NEVER	Specifies that aliases are never dereferenced.	
	LDAP_DEREF_SEARCHING	Specifies that aliases are dereferenced when searching under the base object (but not when finding the base object).	
	LDAP_DEREF_FINDING	Specifies that aliases are dereferenced when finding the base object (but not when searching under the base object).	
	LDAP_DEREF_ALWAYS	Specifies that aliases are always dereferenced when finding the base object and searching under the base object.	
	The data type for the optdata	that should be returned by the server in search results. parameter is (int *). Setting the <i>optdata</i> parameter to by size limit enforced by the client.	

# ldap\_get\_option(3LDAP)

LDAP_OPT_TIMELIMIT Maximum number of seconds that should be spent by the server when answering a search request. The data type for the <i>optdata</i> parameter is (int *). Setting the <i>optdata</i> parameter to LDAP_NO_LIMIT removes any time limit enforced by the client.		
LDAP_OPT_REFERRALS Determines whether or not the client should follow referrals. The data type for the <i>optdata</i> parameter is (int *). <i>optdata</i> can be one of the following values:		
LDAP_OPT_ON	Specifies that the client should follow referrals.	
LDAP_OPT_OFF	Specifies that the client should not follow referrals.	
By default, the client follows ref	errals.	
	O operations are automatically restarted if they abort one of the constants LDAP_OPT_ON or	
LDAP_OPT_PROTOCOL_VERSION Version of the protocol supported by your client. The data type for the <i>optdata</i> parameter is (int *). You can specify either LDAP_VERSION2 or LDAP_VERSION3. If no version is set, the default is LDAP_VERSION2. In order to use LDAP v3 features, you need to set the protocol version to LDAP_VERSION3.		
LDAP_OPT_SERVER_CONTROLS Pointer to an array of LDAPControl structures representing the LDAP v3 server controls you want sent with every request by default. The data type for the <i>optdata</i> parameter for ldap_set_option() is (LDAPControl **) and for ldap_get_option() is (LDAPControl ***).		
LDAP_OPT_CLIENT_CONTROLS Pointer to an array of LDAPControl structures representing the LDAP v3 client controls you want sent with every request by default. The data type for the <i>optdata</i> parameter for ldap_set_option() is (LDAPControl **) and for ldap_get_option() is (LDAPControl ***).		
LDAP_OPT_API_FEATURE_INFO Used to retrieve version information about LDAP API extended features at execution time. The data type for the <i>optdata</i> parameter is (LDAPAPIFeatureInfo *). This option is READ-ONLY and cannot be set.		
data type for the optdata para	LDAP_OPT_HOST_NAME This option sets the host name (or list of hosts) for the primary LDAP server. The data type for the <i>optdata</i> parameter for ldap_set_option() is (char *), and for ldap_get_option() is (char **).	
LDAP_OPT_ERROR_NUMBER The code of the most recent LDAP error that occurred for this session. The data type for the <i>optdata</i> parameter is (int *).		

	LDAP_OPT_ERROR_STRING The message returned with the most recent LDAP error that occurred for this session. The data type for the optdata parameter for ldap_set_option() is (char *) and for ldap_get_option() is (char **).		
LDAP_OPT_MATCHED_DN The matched DN value returned with the most recent LDAP error that occ this session. The data type for the optdata parameter for ldap_set_opt: (char *) and for ldap_get_option() is (char **).			
	LDAP_OPT_REBIND_ARG Lets you set the last argument passed to the routine specified by LDAP_OPT_REBIND_FN. You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the <i>optdata</i> parameter is (void * ).		
	LDAP_OPT_REBIND_FN Lets you set the routine to be called when you need to authenticate a connection with another LDAP server (for example, during the course of a referral). You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the <i>optdata</i> parameter is (LDAP_REBINDPROC_CALLBACK *).		
<b>RETURN VALUES</b>	The ldap_set_option() and	<pre>ldap_get_option() functions return:</pre>	
		f successful	
	_	funsuccessful	
ERRORS	Upon successful completion, both functions return LDAP_SUCCESS, otherwise -1 is returned.		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	
	Stability Level	Evolving	
SEE ALSO	<pre>ldap_init(3LDAP), attribut</pre>	ces(5)	
NOTES	There are other elements in the LDAP structure that you should not change. You should not make any assumptions about the order of elements in the LDAP structure.		

ldap\_get\_values(3LDAP)

NAME	ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, ldap_value_free_len – LDAP attribute value handling functions		
SYNOPSIS	cc[flag] filelldap[library]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	char <b>**ldap_get_values</b> (LDAP <i>*ld</i> , LDAPMessage <i>*entry</i> , char <i>*attr</i> );		
	<pre>struct berval **ldap_get_values_len(LDAP *ld, LDAPMessage *entry,</pre>		
	<pre>ldap_count_values(char **vals);</pre>		
	<pre>ldap_count_values_len(struct ber</pre>	val **vals);	
	<pre>ldap_value_free(char **vals);</pre>		
	<pre>ldap_value_free_len(struct berva)</pre>	1 ** <i>vals</i> );	
DESCRIPTION	These functions are used to retrieve and manipulate attribute values from an LDAP entry as returned by ldap_first_entry(3LDAP) or ldap_next_entry(3LDAP). ldap_get_values() takes the <i>entry</i> and the attribute <i>attr</i> whose values are desired and returns a NULL-terminated array of the attribute's values. <i>attr</i> may be an attribute type as returned from ldap_first_attribute(3LDAP) or ldap_next_attribute(3LDAP), or if the attribute type is known it can simply be given.		
	The number of values in the array can be counted by calling <pre>ldap_count_values(). The array of values returned can be freed by calling <pre>ldap_value_free().</pre></pre>		
	If the attribute values are binary in nature, and thus not suitable to be returned as an array of char *'s, the ldap_get_values_len() function can be used instead. It takes the same parameters as ldap_get_values(), but returns a NULL-terminated array of pointers to berval structures, each containing the length of and a pointer to a value.		
	The number of values in the array can be counted by calling <pre>ldap_count_values_len(). The array of values returned can be freed by calling <pre>ldap_value_free_len().</pre></pre>		
ERRORS	If an error occurs in ldap_get_values() or ldap_get_values_len(), NULL returned and the ld_errno field in the ld parameter is set to indicate the error. See ldap_error(3LDAP) for a description of possible error codes.		
ATTRIBUTES	See attributes(5) for a description of the	following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
	<u> </u>	1J	

338 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

	SUNWldapx (64-bit)
Stability Level	Evolving

# SEE ALSO | ldap(3LDAP), ldap\_first\_entry(3LDAP), ldap\_first\_attribute(3LDAP), ldap\_error(3LDAP)

## **NOTES** These functions allocates memory that the caller must free.

ldap\_memcache(3LDAP)

NAME	ldap_memcache, ldap_memcache_init, ldap_memcache_set, ldap_memcache_get, ldap_memcache_flush, ldap_memcache_destroy, ldap_memcache_update – LDAP client caching functions
SYNOPSIS	cc -flag <i>file</i> lldap [ -library ] #include <ldap.h></ldap.h>
	<pre>int ldap_memcache_init(unsigned long ttl, unsigned long size, char  **baseDNs, struct ldap_thread_fns *thread_fns, LDAPMemCache  **cachep);</pre>
	<pre>int ldap_memcache_set(LDAP *ld, LDAPMemCache **cache);</pre>
	<pre>int ldap_memcache_get(LDAP *ld, LDAPMemCache **cachep);</pre>
	<pre>void ldap_memcache_flush(LDAPMemCache *cache, char *dn, int scope);</pre>
	<pre>void ldap_memcache_destroy(LDAPMemCache *cache);</pre>
	<pre>void ldap_memcache_update(LDAPMemCache *cache);</pre>
DESCRIPTION	Use the ldap_memcache functions to maintain an in-memory client side cache to store search requests. Caching improves performance and reduces network bandwidth when a client makes repeated requests. The <i>cache</i> uses search criteria as the key to the cached items. When you send a search request, the <i>cache</i> checks the search criteria to determine if that request has been previously stored. If the request was stored, the search results are read from the <i>cache</i> .
	Make a call to ldap_memcache_init() to create the in-memory client side <i>cache</i> . The function passes back a pointer to an LDAPMemCache structure, which represents the <i>cache</i> . Make a call to the ldap_memcache_set() function to associate this <i>cache</i> with an LDAP connection handle, an LDAP structure. <i>ttl</i> is the the maximum amount of time (in seconds) that an item can be cached. If a <i>ttl</i> value of 0 is passed, there is no limit to the amount of time that an item can be cached. <i>size</i> is the maximum amount of memory (in bytes) that the cache will consume. A zero value of <i>size</i> means the cache has no size limit. <i>baseDNS</i> is an array of the base DN strings representing the base DNs of the search requests you want cached. If <i>baseDNS</i> is NULL, only the search requests with the specified base DNs will be cached. If <i>baseDNS</i> is NULL, all search requests are cached. The <i>thread_fns</i> parameter takes an ldap_thread_fns structure specifying the functions that you want used to ensure that the cache is thread-safe. You should specify this if you have multiple threads that are using the same connection handle and cache. If <i>you</i> are not using multiple threads, pass NULL for this parameter. Idap_memcache_init() function with an LDAP connection handle. The <i>ld</i> parameter should be the result of a <i>successful</i> call to 1dap_open(3LDAP). The <i>cache</i> parameter should be the result of a <i>cache</i> created by the ldap_memcache_init() call. After you call this function, search requests made over
	the specified LDAP connection will use this cache. To disassociate the cache from the LDAP connection handle, make a call to the ldap_unbind(3LDAP) or

# ldap\_memcache(3LDAP)

		http_momente(022111)
	want to associate a cache with the ldap_memcache_get()	function. Make a call to ldap_memcache_set() if you multiple LDAP connection handles. For example, call function to get the <i>cache</i> associated with one connection, and associate the <i>cache</i> with another connection.
	connection handle (LDAP stru through that connection. Whe	function gets the <i>cache</i> associated with the specified acture). This <i>cache</i> is used by all search requests made in you call this function, the function sets the <i>cachep</i> LDAPMemCache structure that is associated with the
	search request is within the so request is flushed from the <i>cat</i> <i>scope</i> parameter, along with th	clushes search requests from the <i>cache</i> . If the base DN of a cope specified by the $dn$ and <i>scope</i> arguments, the search <i>che</i> . If no DN is specified, the entire cache is flushed. The me $dn$ parameter, identifies the search requests that you this argument can have one of the following values:
	LDAP_SCOPE_BASE LDAP_SCOPE_ONELEVEL LDAP_SCOPE_SUBTREE	
	ldap_memcache_destroy() frees the specified LDAPMemCache structure pointed to by <i>cache</i> from memory. Call this function after you are done working with a <i>cache</i> .	
	removes them. This check is t works. You do not need to cal	checks the cache for items that have expired and ypically done as part of the way the <i>cache</i> normally l this function unless you want to update the <i>cache</i> at this only useful in a multithreaded application, since it will stroyed.
PARAMETERS	ttl	The maximum amount of time (in seconds) that an item can be cached
	size	The maximum amount of memory (in bytes) that the cache will consume.
	baseDNs	An array of the base DN strings representing the base DNs of the search requests you want cached
	thread_fns	A pointer to the ldap_thread_fns structure structure.
	cachep	A pointer to the LDAPMemCache structure
	cache	The result of a <i>cache</i> created by the ldap_memcache_init() call
	ld	The result of a successful call to ldap_open(3LDAP)
	dn	The search requests that you want flushed from the <i>cache</i>

ldap\_memcache(3LDAP)

	scope The search cache	h requests that you want flushed from the	
ERRORS	successful. Otherwise, they return anothe	The functions that have int return values return LDAP_SUCCESS if the operation was uccessful. Otherwise, they return another LDAP error code. See .dap_error(3LDAP) for a list of the LDAP error codes.	
ATTRIBUTES	See attributes(5) for descriptions of the	ne following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWcsl (32-bit)	
		SUNWcslx (64-bit)	
	Interface Stability	Evolving	
SEE ALSO	ldap_error(3LDAP),ldap_open(3LD	AP),ldap_search(3LDAP),attributes(5)	

ldap_modify, ldap_modify_s, ldap_mods_free, ldap_modify_ext, ldap_modify_ext_s – LDAP entry modification functions
cc[ flag ] filelldap[ library ]
<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
<pre>int ldap_modify(LDAP *ld, char *dn, LDAPMod *mods[]);</pre>
<pre>int ldap_modify_s(LDAP *ld, char *dn, LDAPMod *mods[]);</pre>
<pre>void ldap_ mods_ free(LDAPMod **mods, int freemods);</pre>
<pre>int ldap_modify_ext(LDAP *ld, char *dn, LDAPMod **mods, LDAPControl</pre>
<pre>int ldap_modify_ext_s(LDAP *ld, char *dn, LDAPMod **mods, LDAPControl **serverctrls, LDAPControl **clientctrls);</pre>
The function ldap_modify_s() is used to perform an LDAP modify operation. <i>dn</i> is the DN of the entry to modify, and <i>mods</i> is a null-terminated array of modifications to make to the entry. Each element of the <i>mods</i> array is a pointer to an LDAPMod structure, which is defined below.
<pre>typedef struct ldapmod {     int mod_op;     char *mod_type;     union {       char **modv_strvals;       struct berval **modv_bvals;       } mod_vals;     } tDAPMod; #define mod_values mod_vals.modv_bvals #define mod_bvalues mod_vals.modv_bvals</pre>
The <i>mod_op</i> field is used to specify the type of modification to perform and should be one of LDAP_MOD_ADD, LDAP_MOD_DELETE, or LDAP_MOD_REPLACE. The <i>mod_type</i> and <i>mod_values</i> fields specify the attribute type to modify and a null-terminated array of values to add, delete, or replace respectively.
If you need to specify a non-string value (for example, to add a photo or audio attribute value), you should set <i>mod_op</i> to the logical OR of the operation as above (for example, LDAP_MOD_REPLACE) and the constant LDAP_MOD_BVALUES. In this case, <i>mod_bvalues</i> should be used instead of <i>mod_values</i> , and it should point to a null-terminated array of struct bervals, as defined in <lber.h>.</lber.h>
For LDAP_MOD_ADD modifications, the given values are added to the entry, creating the attribute if necessary. For LDAP_MOD_DELETE modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the <i>mod_values</i> field should be set to NULL. For LDAP_MOD_REPLACE modifications, the attribute will have the listed values after the

ldap_modify(3LDAP)	
--------------------	--

modification, having been created if necessary. All modifications are performed in the order in which they are listed.		
ldap_modify_s() returns the LDAP error code resulting from the modify operation.		
The ldap_modify() operation works the same way as ldap_modify_s(), except that it is asynchronous, returning the message id of the request it initiates, or -1 on error. The result of the operation can be obtained by calling ldap_result(3LDAP).		
ldap_mods_free() can be used to free each element of a NULL-terminated array of mod structures. If <i>freemods</i> is non-zero, the <i>mods</i> pointer itself is freed as well.		
The ldap_modify_ext() function initiates an asynchronous modify operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code if not (see ldap_error(3LDAP)). If successful, ldap_modify_ext() places the message id of the request in <i>*msgidp</i> . A subsequent call to ldap_result(3LDAP), can be used to obtain the result of the add request.		
The ldap_modify_ext_s() function initiates a synchronous modify operation and returns the result of the operation itself.		
<pre>ldap_modify_s() returns an ldap error code, either LDAP_SUCCESS or an error (see ldap_error(3LDAP)).</pre>		
<pre>ldap_modify() returns -1 in case of trouble, setting the error field of ld.</pre>		
See attributes(5) for a description of the following attributes:		
ATTRIBUTE TYPE	ATTRIBUTE VALUE	
Availability	SUNWIIdap (32-bit)	
	SUNWldapx (64-bit)	
Stability Level	Evolving	
attributes(5), ldap(3LDAP), lda ldap_get_option(3LDAP)	up_add(3LDAP),ldap_error(3LDAP),	
	order in which they are listed. ldap_modify_s() returns the LDA The ldap_modify() operation wor that it is asynchronous, returning the error. The result of the operation can ldap_mods_free() can be used to mod structures. If <i>freemods</i> is non-zer The ldap_modify_ext() function returns LDAP_SUCCESS if the requese returns a LDAP error code if not (see ldap_modify_ext() places the model call to ldap_result(3LDAP), can be The ldap_modify_ext_s() function returns the result of the operation its ldap_modify_s() returns an ldap ldap_error(3LDAP)). ldap_modify() returns -1 in case See attributes(5) for a description ATTRIBUTE TYPE Availability Stability Level attributes(5), ldap(3LDAP), ldap	

NAME	ldap_modrdn, ldap_modrdn_s, ldap_modrdn2, ldap_modrdn2_s, ldap_rename, ldap_rename_s – modify LDAP entry RDN
SYNOPSIS	cc[flag] file lldap [ library ]
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>
	int <b>ldap_modrdn</b> (LDAP * <i>ld</i> , const char * <i>dn</i> , const char * <i>newrdn</i> );
	<pre>int ldap_modrdn_s(LDAP *ld, const char *dn, const char *newrdn, int</pre>
	<pre>int ldap_modrdn2(LDAP *ld, const char *dn, const char *newrdn, int</pre>
	<pre>int ldap_modrdn2_s(LDAP *ld, const char *dn, const char *newrdn, int</pre>
	<pre>int ldap_rename(LDAP *ld, const char *dn, const char *newrdn, const char *newparent, int deleteoldrdn, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp);</pre>
	<pre>int ldap_rename_s(LDAP *ld, const char *dn, const char *newrdn, const char *newparent, const int deleteoldrdn, LDAPControl **serverctrls, LDAPControl **clientctrls);</pre>
DESCRIPTION	The ldap_modrdn() and ldap_modrdn_s() functions perform an LDAP modify RDN (Relative Distinguished Name) operation. They both take <i>dn</i> , the DN (Distinguished Name) of the entry whose RDN is to be changed, and <i>newrdn</i> , the new RDN, to give the entry. The old RDN of the entry is never kept as an attribute of the entry. ldap_modrdn() is asynchronous. It return the message id of the operation it initiates.ldap_modrdn_s() is synchronous. It returns the LDAP error code that indicates the success or failure of the operation.
	The ldap_modrdn2() and ldap_modrdn2_s() functions also perform an LDAP modify RDN operation. They take the same parameters as above. In addition, they both take the <i>deleteoldrdn</i> parameter ,which is used as a boolean value to indicate whether or not the old RDN values should be deleted from the entry.
	The ldap_rename(), ldap_rename_s() routines are used to change the name, that is, the RDN of an entry. These routines deprecate the ldap_modrdn() and ldap_modrdn_s() routines, as well as ldap_modrdn2() and ldap_modrdn2_s().
	The ldap_rename() and ldap_rename_s() functions both support LDAPv3 server controls and client controls.
ERRORS	The synchronous (_s) versions of these functions return an LDAP error code, either LDAP_SUCCESS or an error. See ldap_error(3LDAP).

#### ldap\_modrdn(3LDAP)

The asynchronous versions return -1 in the event of an error, setting the ld\_errno field of *ld*. See ldap\_error(3LDAP) for more details. Use ldap\_result(3LDAP) to determine a particular unsuccessful result.

ATTRIBUTES See attributes(5) for a description of the following attributes of the ldap\_modrdn(),ldap\_modrdn\_s(),ldap\_modrdn2() and ldap\_modrdn2\_s() functions:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Obsolete

The ldap\_rename() and ldap\_rename\_s() functions have the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Evolving

## **SEE ALSO** | ldap(3LDAP), ldap\_error(3LDAP), attributes(5)

NAME	ldap_open, ldap_init – initialize the LDAP library and open a connection to an LDAP server	
SYNOPSIS	cc[ flag ] filelldap[ library ]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	LDAP <b>*ldap_open</b> (char *host, int port);	
	LDAP <b>*ldap_init</b> (char * <i>host</i> , int <i>port</i> );	
DESCRIPTION	<pre>ldap_open() opens a connection to an LDAP server and allocates an LDAP structure which is used to identify the connection and to maintain per-connection information. ldap_init() allocates an LDAP structure but does not open an initial connection. The ldap_open() function is deprecated and should no longer be used. ldap_init() must be called before any operations are attempted.</pre>	
	<pre>ldap_open() takes host, the hostname on which the LDAP server is running, and port, the port number to which to connect. If the default IANA-assigned port of 389 is desired, LDAP_PORT should be specified for port. The host parameter may contain a blank-separated list of hosts to try to connect to, and each host may optionally by of the form host:port. If present, the :port overrides the port parameter to ldap_open(). Upon successfully making a connection to an LDAP server, ldap_open() returns a pointer to an LDAP structure (opaque structure), which should be passed to subsequent calls to ldap_bind(), ldap_search(), and so forth. Certain fields in the LDAP structure can be set using ldap_set_option(). See ldap_set_option(3LDAP) for more details.</pre>	
	<pre>ldap_init() acts just like ldap_open(), but does not open a connection to the LDAP server. The actual connection open will occur when the first operation is attempted.</pre>	
OPTIONS	Options that affect a particular LDAP instance may be set by calling ldap_set_option(). The settings of these options can be retrieved by calling ldap_get_option().	
	The other supported option is LDAP_OPT_RESTART, which if set will cause the LDAP library to restart the select(1) system call when it is interrupted by the system (that is errno is set to EINTR). This option is not supported on the Macintosh and under MS-DOS.	
	An option can be turned off by clearing the appropriate bit in the ld_options field.	
ERRORS	If an error occurs, these functions will return NULL and errno should be set appropriately.	
ATTRIBUTES	See attributes(5) for a description of the following attributes:	

ldap\_open(3LDAP)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	
	Stability Level	Evolving	
		·	
SEE ALSO	<pre>select(1), errno(3C), ldap(3LDAP), lda attributes(5)</pre>	ap_bind(3LDAP),ldap_option(3LDAP),	
NOTES	There are other elements in the LDAP structure that you should not change. You should not make any assumptions about the order of elements in the LDAP structure.		

NAME	ldap_parse_result, ldap_parse_extended_result, ldap_parse_sasl_bind_result – LDAP message result parser	
SYNOPSIS	cc[ flag ] filelldap[ library ]	
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>int ldap_parse_result(LDAP *ld, LDAPMessage *res, int *errcodep, char</pre>	
	<pre>int ldap_parse_sasl_bind_result(LDAP *ld, LDAPMessage *res, struct</pre>	
	<pre>int ldap_parse_extended_result(LDAP *ld, LDAPMessage *res, char</pre>	
DESCRIPTION	The ldap_parse_extended_result(),ldap_parse_result() and ldap_parse_sasl_bind_result() routines search for a message to parse. These functions skip messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE.	
RETURN VALUES	They return LDAP_SUCCESS if the result was successfully parsed or an LDAP error code if not (see ldap_error(3LDAP)).	
ATTRIBUTES	See attributes(5) for a description of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Availability	SUNWIIdap (32-bit)
		SUNWldapx (64-bit)
	Stability Level	Evolving

**SEE ALSO** ldap\_error(3LDAP), ldap\_result(3LDAP), attributes(5)

# ldap\_result(3LDAP)

NAME	ldap_result, ldap_msgfree – wait for and return LDAP operation result	
SYNOPSIS	<pre>cc[flag] filelldap[ library] #include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>	
	<pre>int ldap_result(LDAP *ld, int msgid, int all, struct timeval *timeout, LDAPMessage **result);</pre>	
	<pre>int ldap_msgfree(LDAPMessage *msg);</pre>	
DESCRIPTION	The ldap_result() function is used to wait for and return the result of an operation previously initiated by one of the LDAP asynchronous operation functions (for example, ldap_search(3LDAP), ldap_modify(3LDAP), etc.). Those functions all return -1 in case of error, and an invocation identifier upon successful initiation of the operation. The invocation identifier is picked by the library and is guaranteed to be unique across the LDAP session. It can be used to request the result of a specific operation from ldap_result() through the <i>msgid</i> parameter.	
	The ldap_result() function will block or not, depending upon the setting of the <i>timeout</i> parameter. If timeout is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a null pointer, the select blocks indefinitely. NU To effect a poll, the timeout argument should be a non-null pointer, pointing to a zero-valued timeval structure. See select(1) for further details.	
	<ul> <li>If the result of a specific operation is required, <i>msgid</i> should be set to the invocation identifier returned when the operation was initiated, otherwise LDAP_RES_ANY should be supplied. The <i>all</i> parameter only has meaning for search responses and is used to select whether a single entry of the search response should be returned, or all results of the search should be returned.</li> <li>A search response is made up of zero or more search entries followed by a search result. If <i>all</i> is set to -, search entries will be returned one at a time as they come in, via separate calls to ldap_result(). If it is set to -1, the search response will only be returned in its entirety, that is, after all entries and the final search result have been received.</li> <li>Upon success, the type of the result received is returned and the <i>result</i> parameter will contain the result of the operation. This result should be passed to the LDAP parsing functions, (see ldap_first_entry(3LDAP)) for interpretation.</li> </ul>	
	<pre>#define LDAP_RES_BIND 0x61L #define LDAP_RES_SEARCH_ENTRY 0x64L #define LDAP_RES_SEARCH_RESULT 0x65L #define LDAP_RES_MODIFY 0x67L #define LDAP_RES_ADD 0x69L #define LDAP_RES_DELETE 0x6bL #define LDAP_RES_MODRDN 0x6dL #define LDAP_RES_COMPARE 0x6fL</pre>	

The ldap\_msgfree() function is used to free the memory allocated for a result by ldap\_result() or ldap\_search\_s(3LDAP) functions. It takes a pointer to the result to be freed and returns the type of the message it freed.

**ERRORS** |ldap\_result() returns -1 if something bad happens, and zero if the timeout specified was exceeded.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWlldap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** select(1), ldap(3LDAP), ldap\_search(3LDAP)

**NOTES** This function allocates memory for results that it receives. The memory can be freed by calling ldap\_msgfree.

# ldap\_search(3LDAP)

NAME	ldap_search, ldap_search_s, ldap_search_ext, ldap_search_ext_s, ldap_search_st – LDAP search operations	
SYNOPSIS	cc[flag] filelldap[library]	
	<pre>#include <sys time.h=""> /* for struct timeval definition */ #include <lber.h> #include <ldap.h></ldap.h></lber.h></sys></pre>	
	<pre>int ldap_search(LDAP *ld, char *base, int scope, char *filter, char *attrs[], int attrsonly);</pre>	
	<pre>int ldap_search_s(LDAP *ld, char *base, int scope, char *filter, char *attrs[], int attrsonly, LDAPMessage **res);</pre>	
	<pre>int ldap_search_st(LDAP *ld, char *base, int scope, char *filter, char *attrs[], int attrsonly, struct timeval *timeout, LDAPMessage **res);</pre>	
	<pre>int ldap_search_ext(LDAP *ld, char *base, int scope, char *filter, char</pre>	
	<pre>int ldap_search_ext_s(LDAP *ld, char *base, int scope, char *filter, char **attrs, int attrsonly, LDAPControl **serverctrls, LDAPControl **clientctrls, struct timeval *timeoutp, int sizelimit);</pre>	
DESCRIPTION	These functions are used to perform LDAP search operations. ldap_search_s() does the search synchronously (that is, not returning until the operation completes). ldap_search_st() does the same, but allows a <i>timeout</i> to be specified. ldap_search() is the asynchronous version, initiating the search and returning the message id of the operation it initiated.	
	<i>Base</i> is the DN of the entry at which to start the search. <i>Scope</i> is the scope of the search and should be one of LDAP_SCOPE_BASE, to search the object itself, LDAP_SCOPE_ONELEVEL, to search the object's immediate children, or LDAP_SCOPE_SUBTREE, to search the object and all its descendents.	
	<i>Filter</i> is a string representation of the filter to apply in the search. Simple filters can be specified as <i>attributetype=attributevalue</i> . More complex filters are specified using a prefix notation according to the following BNF:	
	<pre><filter> ::= '(' <filtercomp> ')' <filtercomp> ::= <and>   <or>   <not>   <simple> <and> ::= '&amp;' <filterlist> <or> ::= ' ' <filterlist> <not> ::= '!' <filters <filterlist=""> ::= <filter>   <filter> <filterlist> ::= <attributetype> <filtertype> <attributevalue> <filtertype> ::= '='   '~='   '&lt;='   '&gt;='</filtertype></attributevalue></filtertype></attributetype></filterlist></filter></filter></filters></not></filterlist></or></filterlist></and></simple></not></or></and></filtercomp></filtercomp></filter></pre>	
	The '~=' construct is used to specify approximate matching. The representation for <attributetype> and <attributevalue> are as described in RFC 1778. In addition,</attributevalue></attributetype>	

	ldap_search(3LDAP)		
	<attributevalue> can be a single * to achiev text and *'s interspersed to achieve substrip</attributevalue>	ve an attribute existence test, or can contain ng matching.	
	For example, the filter "mail=*" will find any entries that have a mail attribute. The filter "mail=*@terminator.rs.itd.umich.edu" will find any entries that have a mail attribute ending in the specified string. To put parentheses in a filter, escape them with a backslash '\' character. See RFC 1588 for a more complete description of allowable filters. See ldap_getfilter(3LDAP) for functions to help in constructing search filters automatically.		
	<i>Attrs</i> is a null-terminated array of attribute types to return from entries that match <i>filter</i> . If NULL is specified, all attributes will be returned. <i>Attrsonly</i> should be set to 1 if only attribute types are wanted. It should be set to 0 if both attributes types and attribute values are wanted.		
	The ldap_search_ext() function initiates an asynchronous search operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code (see ldap_error(3LDAP)). If successful, ldap_search_ext() places the message id of the request in * <i>msgidp</i> . A subsequent call to ldap_result(3LDAP), can be used to obtain the result of the add request.		
	The ldap_search_ext_s() function initiates a synchronous search operation and as such returns the result of the operation itself.		
ERRORS	<pre>ldap_search_s() and ldap_search_st() will return the LDAP error code resulting from the search operation. See ldap_error(3LDAP) for details.</pre>		
	ldap_search() returns -1 when terminating unsuccessfully.		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	
		SUNWldapx (64-bit)	
	Stability Level	Evolving	
SEE ALSO	<pre>ldap(3LDAP), ldap_result(3LDAP), ldap_getfilter(3LDAP), ldap_error(3LDAP)</pre>		
NOTES	Note that both read and list functionality are subsumed by these functions, by using a filter like "objectclass=*" and a scope of LDAP_SCOPE_BASE (to emulate read) or LDAP_SCOPE_ONELEVEL (to emulate list).		
	These functions may allocate memory which must be freed by the calling application. Return values are contained in <ldap.h>.</ldap.h>		

# ldap\_searchprefs(3LDAP)

NAME	ldap_searchprefs, ldap_init_searchprefs, ldap_init_searchprefs_buf, ldap_free_searchprefs, ldap_first_searchobj, ldap_next_searchobj – LDAP search preference configuration routeines
SYNOPSIS	cc[ flag ] filelldap[ library ]
	<pre># include <lber.h> # include <ldap.h></ldap.h></lber.h></pre>
	<pre>int ldap_init_searchprefs(char **file, struct ldap_searchobj</pre>
	<pre>int ldap_init_searchprefs_buf(char **buf, unsigned longlen, struct     ldap_searchobj **solistp);</pre>
	struct ldap_searchobj <b>**ldap_free_searchprefs</b> (struct ldap_searchobj <sup>**</sup> solist);
	struct ldap_searchobj <b>**ldap_first_searchobj</b> (struct ldap_seachobj <sup>**</sup> solist);
	<pre>struct ldap_searchobj **ldap_next_searchobj(struct ldap_seachobj</pre>
DESCRIPTION	These functions provide a standard way to access LDAP search preference configuration data. LDAP search preference configurations are typically used by LDAP client programs to specify which attributes a user may search by, labels for the attributes, and LDAP filters and scopes associated with those searches. Client software presents these choices to a user, who can then specify the type of search to be performed.
	<pre>ldap_init_searchprefs() reads a sequence of search preference configurations from a valid LDAP searchpref configuration file (see ldapsearchprefs.conf(4)). Upon success, 0 is returned and <i>solistp</i> is set to point to a list of search preference data structures.</pre>
	<pre>ldap_init_searchprefs_buf() reads a sequence of search preference configurations from buf (whose size is buflen). buf should point to the data in the format defined for an LDAP search preference configuration file (see ldapsearchprefs.conf(4)). Upon success, 0 is returned and solistp is set to point to a list of search preference data structures.</pre>
	ldap_free_searchprefs() disposes of the data structures allocated by ldap_init_searchprefs().
	<pre>ldap_first_searchpref() returns the first search preference data structure in the list solist. The solist is typically obtained by calling ldap_init_searchprefs().</pre>
	ldap_next_searchpref() returns the search preference after <i>so</i> in the template list <i>solist</i> . <i>A</i> NULL pointer is returned if <i>so</i> is the last entry in the list.
ERRORS	<pre>ldap_init_search_prefs() and ldap_init_search_prefs_bufs() return:</pre>

ldap\_searchprefs(3LDAP)

LDAP_SEARCHPREF_ERR_VERSION	<i>**buf</i> points to data that is newer than can be handled.
LDAP_SEARCHPREF_ERR_MEM	Memory allocation problem.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** | ldap(3LDAP), ldapsearchprefs.conf(4)

Yeong, W., Howes, T., and Hardcastle-Kille, S., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Howes, T., Hardcastle-Kille, S., Yeong, W., and Robbins, C., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Hardcastle-Kille, S., "A String Representation of Distinguished Names", OSI-DS-23, April 1992.

Information Processing - Open Systems Interconnection - The Directory, International Organization for Standardization. International Standard 9594, (1988).

ldap_sort(3LDAP)
------------------

NAME	ldap_sort, ldap_sort_entries, ldap_sort_values, ldap_sort_strcasecmp – LDAP entry sorting functions		
SYNOPSIS	cc[flag] filelldap[library]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>ldap_sort_entries(LDAP *ld, LDAPMessage **chain, char *attr, int</pre>		
	<code>ldap_sort_values(LDAP *<math>ld</math>, char **a</code>	<pre>vals, int (*cmp)());</pre>	
	<pre>ldap_sort_strcasecmp(char *a, cha</pre>	r *b);	
DESCRIPTION	These functions are used to sort lists of entries and values retrieved from an LDAP server. ldap_sort_entries() is used to sort a chain of entries retrieved from an LDAP search call either by DN or by some arbitrary attribute in the entries. It takes ld, the LDAP structure, which is only used for error reporting, <i>chain</i> , the list of entries as returned by ldap_search_s(3LDAP) or ldap_result(3LDAP). <i>attr</i> is the attribute to use as a key in the sort or NULL to sort by DN, and cmp is the comparison function to use when comparing values (or individual DN components if sorting by DN). In this case, cmp should be a function taking two single values of the <i>attr</i> to sort by, and returning a value less than zero, equal to zero, or greater than zero, depending on whether the first argument is less than, equal to, or greater than the second argument. The convention is the same as used by qsort(3C), which is called to do the actual sorting.		
	<pre>ldap_sort_values() is used to sort an array of values from an entry, as returned by ldap_get_values(3LDAP). It takes the LDAP connection structure ld, the array of values to sort vals, and cmp, the comparison function to use during the sort. Note that cmp will be passed a pointer to each element in the vals array, so if you pass the normal char ** for this parameter, cmp should take two char **'s as arguments (that is, you cannot pass strcasecmp or its friends for cmp). You can, however, pass the function ldap_sort_strcasecmp() for this purpose.</pre>		
	For example:		
	LDAP *ld; LDAPMessage *res; /* call to ldap_search_s( ), fill in res, retrieve sn attr */		
	<pre>/* now sort the entries on surname attribute */ if ( ldap_sort_entries( ld, &amp;res, "sn", ldap_sort_strcasecmp ) != 0 )     ldap_perror( ld, "ldap_sort_entries" );</pre>		
ATTRIBUTES	See attributes(5) for a description of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWIIdap (32-bit)	

356 man pages section 3: Networking Library Functions • Last Revised 25 May 1998

ldap\_sort(3LDAP)

	SUNWldapx (64-bit)
Stability Level	Evolving

**SEE ALSO** 

**O** | ldap(3LDAP), ldap\_search(3LDAP), ldap\_result(3LDAP), qsort(3C)

**NOTES** The ldap\_sort\_entries() function applies the comparison function to each value of the attribute in the array as returned by a call to ldap\_get\_values(3LDAP), until a mismatch is found. This works fine for single-valued attributes, but may produce unexpected results for multi-valued attributes. When sorting by DN, the comparison function is applied to an exploded version of the DN, without types. The return values for all of these functions are declared in the <ldap.h> header file. Some functions may allocate memory which must be freed by the calling application.

# ldap\_ufn(3LDAP)

NAME	ldap_ufn, ldap_ufn_search_s, ldap_ufn_search_c, ldap_ufn_search_ct, ldap_ufn_setfilter, ldap_ufn_setprefix, ldap_ufn_timeout – LDAP user friendly search functions			
SYNOPSIS	cc[ flag ] filelldap[ library ]			
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>			
	<pre>int ldap_ufn_search_c(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res, int (*cancelproc)(), void *cancelparm);</pre>			
	<pre>int ldap_ufn_search_ct(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res, int (*cancelproc)(), void *cancelparm, char *tag1, char *tag2, char *tag3);</pre>			
	<pre>int ldap_ufn_search_s(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res);</pre>			
	LDAPFiltDesc *ldap_ufn_setfilter(LDAP *ld, char *fname);			
	<pre>void ldap_ufn_setprefix(LDAP *ld, char *prefix);</pre>			
	<pre>int ldap_ufn_timeout (void *tvparam);</pre>			
DESCRIPTION	These functions are used to perform LDAP user friendly search operations. ldap_ufn_search_s() is the simplest form. It does the search synchronously. It takes ld to identify the the LDAP connection. The <i>ufn</i> parameter is the user friendly name for which to search. The <i>attrs, attrsonly</i> and <i>res</i> parameters are the same as for ldap_search(3LDAP).			
	The ldap_ufn_search_c() function functions the same as ldap_ufn_search_s(), except that it takes <i>cancelproc</i> , a function to call periodicly during the search. It should be a function taking a single void * argument, given by <i>calcelparm</i> . If <i>cancelproc</i> returns a non-zero result, the search will be abandoned and no results returned. The purpose of this function is to provide a way for the search to be cancelled, for example, by a user or because some other condition occurs.			
	The ldap_ufn_search_ct() function is like ldap_ufn_search_c(), except that it takes three extra parameters. <i>tag1</i> is passed to the ldap_init_getfilter(3LDAP) function when resolving the first component of the UFN. <i>tag2</i> is used when resolving intermediate components. <i>tag3</i> is used when resolving the last component. By default, the tags used by the other UFN search functions during these three phases of the search are "ufn first", "ufn intermediate", and "ufn last".			
	The ldap_ufn_setfilter() function is used to set the ldapfilter.conf(4) file for use with the ldap_init_getfilter(3LDAP) function to <i>fname</i> .			
	The ldap_ufn_setprefix() function is used to set the default prefix (actually, it's a suffix) appended to UFNs before searhing. UFNs with fewer than three components have the prefix appended first, before searching. If that fails, the UFN is tried with progressively shorter versions of the prefix, stripping off components. If the UFN has			

three or more components, it is tried by itself first. If that fails, a similar process is applied with the prefix appended.

The ldap\_ufn\_timeout() function is used to set the timeout associated with ldap\_ufn\_search\_s() searches. The *timeout* parameter should actually be a pointer to a struct timeval (this is so ldap\_ufn\_timeout() can be used as a cancelproc in the above functions).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

#### SEE ALSO gettimeofday(3C), ldap(3LDAP), ldap\_search(3LDAP), ldap\_getfilter(3LDAP), ldapfilter.conf(4), ldap\_error(3LDAP)

**NOTES** These functions may allocates memory. Return values are contained in <ldap.h>.

# ldap\_url(3LDAP)

NAME	ldap_url, ldap_is_ldap_url, ldap_url_parse, ldap_free_urldesc, ldap_url_search, ldap_url_search_s, ldap_url_search_st, ldap_dns_to_url, ldap_dn_to_url – LDAP Uniform Resource Locator functions		
SYNOPSIS	cc[ flag ] filelldap[ library ]		
	<pre>#include <lber.h> #include <ldap.h></ldap.h></lber.h></pre>		
	<pre>int ldap_is_ldap_url (char *url);</pre>		
	<pre>int ldap_url_parse(char *url, LDAPURLDesc **ludpp);</pre>		
	<pre>ldap_free_urldesc(LDAPURLDesc *ludp);</pre>		
	<pre>int ldap_url_search(LDAP *ld, char *url, int attrsonly);</pre>		
	<pre>int ldap_url_search_s(LDAP *ld, char *url, int attrsonly, LDAPMessage</pre>		
	<pre>int ldap_url_search_st(LDAP *ld, char *url, int attrsonly, struct timeval *timeout, LDAPMessage **res);</pre>		
	<pre>char *ldap_dns_to_url(LDAP *ld, char *dns_name, char *attrs, char *scope, char *filter);</pre>		
	<pre>char *ldap_dn_to_url(LDAP *ld, char *dn, int nameparts);</pre>		
DESCRIPTION	These functions support the use of LDAP URLs (Uniform Resource Locators). LDA URLs look like this:		
	<pre>ldap://hostport/dn[?attributes[?scope[?filter]]] where:</pre>		
	hostport	Host name with an optional ":portnumber".	
	dn	Base DN to be used for an LDAP search operation.	
	attributes	Comma separated list of attributes to be retrieved.	
	scope	One of these three strings: base one sub (default=base).	
	filter	LDAP search filter as used in a call to ldap_search(3LDAP).	
	Here is an example:		
		mich.edu/c=US?o,description?one?o=umich pped in angle-brackets and/or preceded by "URL:" are also	
	opposed to some c	arl() returns a non-zero value if <i>url</i> looks like an LDAP URL (as other kind of URL). It can be used as a quick check for an LDAP cl_parse() function should be used if a more thorough check is	

	<pre>ldap_url_parse() breaks down an LDAP URL passed in url into its component pieces. If successful, zero is returned, an LDAP URL description is allocated, filled in, and ludpp is set to point to it. See RETURN VALUES (below) for values returned upon error.</pre>		
	<pre>ldap_free_urldesc() should be called to free an LDAP URL description that was obtained from a call to ldap_url_parse().</pre>		
	<pre>ldap_url_search() initiates an asynchronous LDAP search based on the contents of the url string. This function acts just like ldap_search(3LDAP) except that many search parameters are pulled out of the URL.</pre>		
	<pre>ldap_url_search_s() performs a synchronous LDAP search based on the contents of the url string. This function acts just like ldap_search_s(3LDAP) except that many search parameters are pulled out of the URL.</pre>		
	<pre>ldap_url_search_st() performs a synchronous LDAP URL search with a specified timeout. This function acts just like ldap_search_st(3LDAP) except that many search parameters are pulled out of the URL.</pre>		
	ldap_dns_to_url() locates the LDAP URL associated with a DNS domain name. The supplied DNS domain name is converted into a distinguished name. The directory entry specified by that distinguished name is searched for a labeledURI attribute. If successful then the corresponding LDAP URL is returned. If unsuccessful then that entry's parent is searched and so on until the target distinguished name is reduced to only two nameparts. If <i>dns_name</i> is NULL then the environment variable LOCALDOMAIN is used. If <i>attrs</i> is not NULL then it is appended to the URL's attribute list. If <i>scope</i> is not NULL then it overrides the URL's scope. If <i>filter</i> is not NULL then it is merged with the URL's filter. If an error is encountered then zero is returned, otherwise a string URL is returned. The caller should free the returned string if it is non-zero.		
	<pre>ldap_dn_to_url() locates the LDAP URL associated with a distinguished name. The number of nameparts in the supplied distinguished name must be provided. The specified directory entry is searched for a labeledURI attribute. If successful then the LDAP URL is returned. If unsuccessful then that entry's parent is searched and so on until the target distinguished name is reduced to only two nameparts. If an error is encountered then zero is returned, otherwise a string URL is returned. The caller should free the returned string if it is non-zero.</pre>		
<b>RETURN VALUES</b>	<b>S</b> Upon error, one of these values is returned for ldap_url_parse():		
	LDAP_URL_ERR_NOTLDAP	URL doesn't begin with "ldap://".	
	LDAP_URL_ERR_NODN	URL has no DN (required).	
	LDAP_URL_ERR_BADSCOPE	URL scope string is invalid.	
	LDAP_URL_ERR_MEM	Can't allocate memory space.	
ATTRIBUTES	See attributes(5) for a description of the following attributes:		

ldap\_url(3LDAP)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWIIdap (32-bit)
	SUNWldapx (64-bit)
Stability Level	Evolving

## **SEE ALSO** ldap(3LDAP), ldap\_search(3LDAP)

An LDAP URL Format, Tim Howes and Mark Smith, December 1995. Internet Draft (work in progress). Currently available at this URL:

ftp://ds.internic.net/internet-drafts/draft-ietf-asid-ldap-format-03.txt

NAME	ldap_version – get version information about the LDAP SDK for C	
SYNOPSIS	cc -flag <i>file</i> lldap [ -library ] #include <ldap.h></ldap.h>	
	<pre>int ldap_version(LDAPVERSION *ver);</pre>	
DESCRIPTION	A call to this function returns the version information for the LDAP SDK for C. This is a deprecated function. Use ldap_get_option(LDAP) instead. The version information is returned in the LDAPVersion structure pointed to by <i>ver</i> . If NULL is passed for <i>ver</i> , then only the SDK version will be returned.	
RETURN VALUES	The ldap_version() function returns the version number of the LDAP SDK for C, multiplied by 100. For example, for version 1.0 of the LDAP SDK for C, the function returns 100.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Obsolete

**SEE ALSO** ldap\_get\_option(LDAP), attributes(5)

listen(3SOCKET)	
-----------------	--

NAME	listen – listen for connections on a socket		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>int listen(int s, int backlog);</pre>		
DESCRIPTION	To accept connections, a socket is first created with socket(3SOCKET), a backlog for incoming connections is specified with listen() and then the connections are accepted with accept(3SOCKET). The listen() call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.		
	The <i>backlog</i> parameter defines the maximum length the queue of pending connections may grow to.		
	If a connection request arrives with the queue full, the client will receive an error with an indication of ECONNREFUSED for AF_UNIX sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For AF_INET and AF_INET6sockets, the TCP will retry the connection. If the <i>backlog</i> is not cleared by the time the tcp times out, the connect will fail with ETIMEDOUT.		
<b>RETURN VALUES</b>	A 0 return value indicates success; –1 indicates an error.		
ERRORS	The call fails if:		
	EBADFThe argument s is not a valid file descriptor.		
	ENOTSOCK	The argume	ent <i>s</i> is not a socket.
	EOPNOTSUPP	The socket i listen().	s not of a type that supports the operation
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	MT-Level		Safe
SEE ALSO	accept(3SOCKET), connect(3SOCKET), socket(3SOCKET), attributes(5), socket(3HEAD)		
NOTES	There is currently no <i>backlog</i> limit.		

NAME	listen – listen for socket connections and limit the queue of incoming connections		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	<pre>int listen(int socket, int backlog);</pre>		
DESCRIPTION	The listen() function marks a connection-mode socket, specified by the <i>socket</i> argument, as accepting connections, and limits the number of outstanding connections in the socket's listen queue to the value specified by the <i>backlog</i> argument.		
	If listen() is called with a <i>backlog</i> argument value that is less than 0, the function sets the length of the socket's listen queue to 0.		
	The implementation may include incomplete connections in the queue subject to the queue limit. The implementation may also increase the specified queue limit internally if it includes such incomplete connections in the queue subject to this limit.		
	Implementations may limit the length of the socket's listen queue. If <i>backlog</i> exceeds the implementation-dependent maximum queue length, the length of the socket's listen queue will be set to the maximum supported value.		
	The socket in use may require the process to have appropriate privileges to use the listen() function.		
<b>RETURN VALUES</b>	Upon successful completions, listen() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The listen() function will fail if:		
	EBADF	The socket argument is not a valid file descriptor.	
	EDESTADDRREQ	The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.	
	EINVAL	The <i>socket</i> is already connected.	
	ENOTSOCK	The <i>socket</i> argument does not refer to a socket.	
	EOPNOTSUPP	The socket protocol does not support listen().	
	The listen() function may	fail if:	
	EACCES	The calling process does not have the appropriate privileges.	
	EINVAL	The <i>socket</i> has been shut down.	
	ENOBUFS	Insufficient resources are available in the system to complete the call.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		

# listen(3XNET)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO accept(3XNET), connect(3XNET), socket(3XNET), attributes(5)

366 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

NAME	netdir, netdir_getbyname, netdir_getbyaddr, netdir_free, netdir_options, taddr2uaddr, uaddr2taddr, netdir_perror, netdir_sperror, netdir_mergeaddr – generic transport name-to-address translation			
SYNOPSIS	<pre>#include <netdir.h></netdir.h></pre>			
	<pre>int netdir_getbyname(const struct netconfig *config, const struct nd_hostserv *service, struct nd_addrlist **addrs);</pre>			
	<pre>int netdir_getbyaddr(const struct netconfig *config, struct nd_hostservlist **service, const struct netbuf *netaddr);</pre>			
	<pre>void netdir_free(void *ptr, const int struct_type);</pre>			
	<pre>int netdir_options(const struct netconfig *config, const int option,</pre>			
	<pre>char *taddr2uaddr(const struct netconfig *config, const struct netbuf *addr);</pre>			
	<pre>struct netbuf *uaddr2taddr(const struct netconfig *config, const char *uaddr);</pre>			
	<pre>void netdir_perror(char *s);</pre>			
	<pre>char *netdir_sperror(void);</pre>			
DESCRIPTION	These routines provide a generic interface for name-to-address mapping that will work with all transport protocols. This interface provides a generic way for programs to convert transport specific addresses into common structures and back again. The netconfig structure, described on the netconfig(4) manual page, identifies the transport.			
	The netdir_getbyname() routine maps the machine name and service name in the nd_hostserv structure to a collection of addresses of the type understood by the transport identified in the netconfig structure. This routine returns all addresses that are valid for that transport in the nd_addrlist structure. The nd_hostserv structure contains the following members:			
	<pre>char /* host name */ *h_serv; /* service name */</pre>			
	The nd_addrlist structure contains the following members:			
	<pre>int n_cnt;  /* number of addresses */ struct netbuf *n_addrs;</pre>			
	<pre>netdir_getbyname() accepts some special-case host names. The host names are defined in <netdir.h>. The currently defined host names are:</netdir.h></pre>			
	HOST_SELFRepresents the address to which local programs will bind their endpoints. HOST_SELF differs from the host			

## netdir(3NSL)

	name provided by gethostname(3C), which represents the address to which <i>remote</i> programs will bind their endpoints.	
HOST_ANY	Represents any host accessible by this transport provider. HOST_ANY allows applications to specify a required service without specifying a particular host name.	
HOST_SELF_CONNECT	Represents the host address that can be used to connect to the local host.	
HOST_BROADCAST	Represents the address for all hosts accessible by this transport provider. Network requests to this address will be received by all machines.	
All fields of the nd_hostser	v structure must be initialized.	
To find the address of a given host and service on all available transports, call the netdir_getbyname() routine with each struct netconfig structure returned by getnetconfig(3NSL).		
The netdir_getbyaddr() routine maps addresses to service names. This routine returns <i>service</i> , a list of host and service pairs that would yield this address. If more than one tuple of host and service name is returned, then the first tuple contains the preferred host and service names:		
<pre>struct nd_hostservlist {     int *h_cnt;</pre>		
The netdir_free() structure is used to free the structures allocated by the name to address translation routines. <i>ptr</i> points to the structure that has to be freed. The struct_type identifies the structure:		
struct nd_addrlist ND_ struct hostserv ND_	ADDR ADDRLIST HOSTSERV HOSTSERVLIST	
The universal address returned by $taddr2uaddr()$ should be freed by free().		
The netdir_options() routine is used to do all transport-specific setups and option management. <i>fildes</i> is the associated file descriptor. <i>option, fildes,</i> and <i>pointer_to_args</i> are passed to the netdir_options() routine for the transport specified in <i>config.</i> Currently four values are defined for <i>option</i> :		
ND_SET_BROADCAST		

ND\_SET\_RESERVEDPORT ND\_CHECK\_RESERVEDPORT ND\_MERGEADDR

The taddr2uaddr() and uaddr2taddr() routines support translation between universal addresses and TLI type netbufs. The taddr2uaddr() routine takes a struct netbuf data structure and returns a pointer to a string that contains the universal address. It returns NULL if the conversion is not possible. This is not a fatal condition as some transports may not suppose a universal address form.

uaddr2taddr() is the reverse of taddr2uaddr(). It returns the struct netbuf data structure for the given universal address.

If a transport provider does not support an option, netdir\_options returns -1 and the error message can be printed through netdir\_perror() or netdir sperror().

The specific actions of each option follow.

ND SET BROADCAST

Sets the transport provider up to allow broadcast, if the transport supports broadcast. *fildes* is a file descriptor into the transport (i.e., the result of a t\_open of /dev/udp). *pointer\_to\_args* is not used. If this completes, broadcast operations may be performed on file descriptor *fildes*.

ND SET RESERVEDPORT

Allows the application to bind to a reserved port, if that concept exists for the transport provider. *fildes* is an unbound file descriptor into the transport. If *pointer\_to\_args* is NULL, *fildes* will be bound to a reserved port. If *pointer\_to\_args* is a pointer to a netbuf structure, an attempt will be made to bind to any reserved port on the specified address.

ND CHECK RESERVEDPORT

Used to verify that the address corresponds to a reserved port, if that concept exists for the transport provider. *fildes* is not used. *pointer\_to\_args* is a pointer to a netbuf structure that contains the address. This option returns 0 only if the address specified in *pointer\_to\_args* is reserved.

ND MERGEADDR

USED TO TAKE A "LOCAL ADDRESS" (LIKE THE 0.0.0.0 ADDRESS THAT TCP USES) AND RETURN A "REAL ADDRESS" THAT CLIENT MACHINES CAN CONNECT TO. *FILDES* IS NOT USED. *POINTER\_TO\_ARGS* IS A POINTER TO A STRUCT ND\_MERGEARG, WHICH HAS THE FOLLOWING MEMBERS:

```
char s_uaddr; /* server's universal address */
char c_uaddr; /* client's universal address */
char m_uaddr; /* the result */
```

If s\_uaddr is something like 0.0.0.1.12, and, if the call is successful, m\_uaddr will be set to something like 192.11.109.89.1.12. For most transports, m\_uaddr is exactly what s\_uaddr is.

netdir(3NSL)			
RETURN VALUES	The netdir_perror() routine prints an error message on the standard output stating why one of the name-to-address mapping routines failed. The error message is preceded by the string given as an argument.		
	The netdir_sperror() routine returns a string containing an error message stating why one of the name-to-address mapping routines failed.		
	netdir_sperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	<pre>gethostname(3C), getnetconfig(3NSL) attributes(5)</pre>	), getnetpath(SINSL), netconrig(4),	

NAME	nis_error, nis_sperrno, nis_perror, nis_lerror, nis_sperror, nis_sperror_r – display NIS+ error messages		
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>		
	char <b>*nis_sperrno</b> (nis_error <i>status</i> )	;	
	void <b>nis_perror</b> (nis_error status, c	char *label);	
	void <b>nis_lerror</b> (nis_error status, o	char *label);	
	<pre>char *nis_sperror_r(nis_error status, char *label, char *buf,</pre>		
	char <b>*nis_sperror</b> (nis_error <i>status</i> )	, char *label);	
DESCRIPTION	TION These functions convert NIS+ status values into text strings.		
	nis_sperrno() simply returns a pointer to a string constant which is the error string.		
	nis_perror() prints the error message corresponding to <i>status</i> as " <i>label</i> : error message" on standard error.		
	nis_lerror() sends the error text to $syslog(3C)$ at level LOG_ERR.		
	The function nis_sperror_r(), returns a copied using the strdup() function (See s string buffer, <i>buf</i> , large enough to hold the e guaranteed to be sufficiently large). <i>status</i> an nis_perror(). The pointer returned by n is, the pointer returned by the function is a of characters to copy from the error string to	<pre>tring(3C)). The caller must supply a error string (a buffer size of 128 bytes is nd label are the same as for is_sperror_r() is the same as buf, that pointer to buf. length specifies the number</pre>	
	The last function, nis_sperror(), is similar to nis_sperror_r() except that the string is returned as a pointer to a buffer that is reused on each call. nis_sperror_r() is the preferred interface, since it is suitable for single-threaded and multi-threaded programs.		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level	Safe	

**SEE ALSO** niserror(1), string(3C), syslog(3C), attributes(5)

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

nis\_groups(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 [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>
	<pre>bool_t nis_ismember(nis_name principal, nis_name group);</pre>
	<pre>nis_error nis_addmember(nis_name member, nis_name group);</pre>
	<pre>nis_error nis_removemember(nis_name member, nis_name group);</pre>
	<pre>nis_error nis_creategroup(nis_name group, uint_t flags);</pre>
	<pre>nis_error nis_destroygroup(nis_name group);</pre>
	<pre>void nis_print_group_entry(nis_name group);</pre>
	<pre>nis_error nis_verifygroup(nis_name group);</pre>
DESCRIPTION	These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.
	The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.
	There are three types of group members:
	<ul> <li>An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz."</li> </ul>
	<ul> <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> </ul>
	<ul> <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>
	Any member may be made <i>negative</i> by prefixing it with a minus sign $('-')$ . A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.
	A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.
	The nis_ismember() function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i> ; otherwise it returns FALSE.

The nis\_addmember() and nis\_removemember() functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question.

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.

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 callbool\_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:

Networking Library Functions 373

# nis\_groups(3NSL)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	<pre>nisgrpadm(1), nis_objects(3NSL), att;</pre>	ributes(5)
NOTES	These functions only accept fully-qualified	NIS+ names.
	A group is represented by a NIS+ object (se part that is defined in the group_obj struc	
	<pre>uint_t gr_flags; /* Interpretation</pre>	Flags
	<pre>uint_t gr_members_len; nis_name *gr_members_val; } gr_members; /* Array of members</pre>	*/
	NIS+ servers and clients maintain a local ca performance when checking for group men group change, servers and clients with that either the group cache has expired or it is ex flushed programmatically by calling the ni TAG_GCACHE and a value of 1.	nbership. Should the membership of a group cached will not see the change until xplicitly flushed. A server's cache may be
	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_local_names, nis_local_directory, nis_local_host, nis_local_group, nis_local_principal – NIS+ local names		
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>		
	<pre>nis_name nis_local_directory(void);</pre>		
	<pre>nis_name nis_local_host(void);</pre>		
	<pre>nis_name nis_local_group(void);</pre>		
	nis_name <b>nis_local_principal</b> (voi	d);	
DESCRIPTION	These functions return several default NIS- process.	+ names associated with the current	
	nis_local_directory() returns the name of the NIS+ domain for this machine. This is currently the same as the Secure RPC domain returned by the sysinfo(2) system call.		
	<pre>nis_local_host() returns the NIS+ name of the current machine. This is the fully qualified name for the host and is either the value returned by the gethostname(3C) function or, if the host name is only partially qualified, the concatenation of that value and the name of the NIS+ directory. Note that if a machine's name and address cannot be found in the local NIS+ directory, its hostname must be fully qualified.</pre>		
	nis_local_group() returns the name of the current NIS+ group name. This is currently set by setting the environment variable NIS_GROUP to the groupname.		
	nis_local_principal() returns the NIS+ principal name for the user associated with the effective UID of the calling process. This function maps the effective uid into a principal name by looking for a LOCAL type credential (see nisaddcred(1M)) in the table named <i>cred.org_dir</i> in the default domain.		
	Note: The result returned by these routines is a pointer to a data structure with the NIS+ library, and should be considered a "read-only" result and should not be modified.		
ENVIRONMENT VARIABLES	NIS_GROUP This variable contains the name of the local NIS+ group. If the name is not fully qualified, the value returned by nis_local_directory() will be concatenated to it.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
		·	

nis\_local\_names(3NSL)

nis_names, nis_lookup, nis_ac namespace functions	ld, nis_remove, nis_modify, nis_freeresult – NIS+
<pre>cc [ flag ] filelnsl #include <rpcsvc nis.h=""></rpcsvc></pre>	[ library ]
nis_result <b>*nis_lookup</b>	(nis_name <i>name</i> , uint_t <i>flags</i> );
nis_result <b>*nis_add</b> (nis	<pre>s_name name, nis_object *obj);</pre>
<pre>nis_result *nis_remove</pre>	(nis_name <i>name</i> , nis_object * <i>obj</i> );
nis_result * <b>nis_modify</b>	<pre>(nis_name name, nis_object *obj);</pre>
void <b>nis_freeresult</b> (nis	s_result *result);
	cate and manipulate all NIS+ objects (see the NIS+ entry objects. To look up the NIS+ entry fer to nis_subr(3NSL).
server.nis_add() and nis_:	S+ name and returns a copy of that object from a NIS+ remove() add and remove objects to the NIS+ _modify() can change specific attributes of an object espace.
	d only with names that refer to an NIS+ Directory, NIS+ rivate object. If a name refers to an NIS+ entry object, the 3NSL) should be used.
<pre>function must be called to free nis_lookup(), nis_add() pointer to a nis_result str nis_freeresult() when ye</pre>	<pre>memory associated with a nis_result structure. This the memory associated with a NIS+ result. ), nis_remove(), and nis_modify() all return a ructure which <i>must</i> be freed by calling bu have finished using it. If one or more of the objects to be retained, they can be copied with see nis_subr(3NSL)).</pre>
and a flags parameter, <i>flags</i> , w correspond to the syntax of a mis_lookup() function is the qualified name. If the parameter	rameters, the name of the object to be resolved in <i>name</i> , hich is defined below. The object name is expected to non-indexed NIS+ name (see nis_tables(3NSL)). The e <i>only</i> function from this group that can use a non-fully ter <i>name</i> is not a fully qualified name, then the flag ed in the call. If this flag is not specified, the function ADNAME.
The <i>flags</i> parameter is construct following list.	cted by logically ORing zero or more flags from the
FOLLOW_LINKS	When specified, the client library will "follow" links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this
	<pre>namespace functions cc [ flag ] filelnsl #include <rpcsvc nis.h=""> nis_result *nis_lookup nis_result *nis_add(nis nis_result *nis_modify void nis_freeresult(nis These functions are used to loo nis_objects(3NSL)) except objects within a NIS+ table, re nis_lookup() resolves a NII server. nis_add() and nis namespace, respectively. nis that already exists in the name These functions should be use Table, NIS+ Group, or NIS+ Pf functions listed in nis_subr() nis_freeresult() frees all function must be called to free nis_lookup(), nis_add() pointer to a nis_result str nis_freeresult() when yo returned in the structure need nis_clone_object(3NSL)( nis_lookup() takes two par and a flags parameter, flags, w correspond to the syntax of a nis_lookup() function is th qualified name. If the paramete EXPAND_NAME must be specifi will fail with the error NIS_BA The flags parameter is construct following list.</rpcsvc></pre>

nis\_names(3NSL)

process will iterate until the either a object is found that is not a LINK 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. When specified, the client library will bypass any object MASTER ONLY caches and any domain replicas and fetch the object from the NIS+ master server for the object's domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant. EXPAND NAME When specified, the client library will attempt to expand a partially qualified name by calling the function nis getnames() (see nis subr(3NSL)) which uses the environment variable NIS PATH. 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\_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.

zticks	The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
dticks	The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If

nis_names(3NSL)			
		the request results in a of all the time spent in	multiple calls to the database, this is the sum n those calls.
	aticks		"accelerators" or caches. This includes the e the server needed to resolve the request.
	cticks	enter the client library subtracting the sum o	n the request. This clock starts when you and stops when a result is returned. By f the other ticks values from this value, you verhead of generating a NIS+ request.
	service code itself.	Subtracting the sum of the time spent in the cli	alue in <i>zticks</i> will yield the time spent in the the values in <i>zticks</i> and <i>aticks</i> from the value ient library itself. Note: all of the tick times
RETURN VALUES	The client library c salient ones are do	-	rror returns and diagnostics. The more
	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_CACHEEXPIR	ED	The object returned came from an object cache taht has <i>expired</i> . The time to live value has gone to zero and the object may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.
	NIS_NAMEUNREAC	HABLE	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	T	The object returned is of an unknown type.
	NIS_TRYAGAIN		The server connected to was too busy to handle your request. For the <i>add</i> , <i>remove</i> , and <i>modify</i> operations this is returned when either the master server for a directory is unavailable or it is in the process of

## nis\_names(3NSL)

	checkpointing its database. It can also be returned when the server is updating it's internal state. And in the case of nis_list() if the client specifies a callback and the server does not have enough resources to handle the callback.
NIS_SYSTEMERROR	A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.
NIS_NOT_ME	A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.
NIS_NOMEMORY	Generally a fatal result. It means that the service ran out of heap space.
NIS_NAMEEXISTS	An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.
NIS_NOTMASTER	An attempt was made to update the database on a replica server.
NIS_INVALIDOBJ	The object pointed to by <i>obj</i> is not a valid NIS+ object.
NIS_BADNAME	The name passed to the function is not a legal NIS+ name.
NIS_LINKNAMEERROR	The name passed resolved to a <i>LINK</i> type object and the contents of the link pointed to an invalid name.
NIS_NOTSAMEOBJ	An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.
NIS_NOSUCHNAME	This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table,

nis\_names(3NSL)

		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.
		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.
		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.
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	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	ATTRIBUTE TYPE MT-Level	ATTRIBUTE VALUE MT-Safe
SEE ALSO		MT-Safe
SEE ALSO NOTES	MT-Level nis_error(3NSL), nis_objects(3NSL), nis_tables(3NSL), attributes(5)	MT-Safe
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to
	MT-Level nis_error(3NSL), nis_objects(3NSL) nis_tables(3NSL), attributes(5) You cannot modify the name of an object in reside in a different domain.	MT-Safe , nis_server(3NSL), nis_subr(3NSL), f that modification would cause the object to

**NAME** | nis\_objects – NIS+ object formats

**SYNOPSIS** 

cc [ flag ... ] file ... -lnsl [ library ... ]
/usr/include/rpcsvc/nis\_objects.x

Common Attributes The NIS+ service uses a variant record structure to hold the contents of the objects that are used by the NIS+ service. These objects all share a common structure which defines a set of attributes that all objects possess. The nis\_object structure contains the following members:

```
typedef char *nis_name;
struct nis_object {
    nis_oid zo_oid;
    nis_name zo_name;
    nis_name zo_owner;
    nis_name zo_group;
    nis_name zo_domain;
    uint_t zo_access;
    uint32_t zo_ttl;
    objdata zo_data;
    };
```

In this structure, the first member zo\_oid, is a 64 bit number that uniquely identifies this instance of the object on this server. This member is filled in by the server when the object is created and changed by the server when the object is modified. When used in conjunction with the object's name and domain it uniquely identifies the object in the entire NIS+ namespace.

The second member, zo\_name, contains the leaf name of the object. This name is *never* terminated with a '.' (dot). When an object is created or added to the namespace, the client library will automatically fill in this field and the domain name from the name that was passed to the function.

zo\_domain contains the name of the NIS+ domain to which this object belongs. This
information is useful when tracking the parentage of an object from a cache. When
used in conjunction with the members zo\_name and zo\_oid, it uniquely identifies an
object. This makes it possible to always reconstruct the name of an object by using the
code fragment

sprintf(buf,"%s.%s", obj->zo\_name, obj->zo\_domain);

The zo\_owner and zo\_group members contain the NIS+ names of the object's principal owner and group owner, respectively. Both names *must be* NIS+ fully qualified names. However, neither name can be used directly to identify the object they represent. This stems from the condition that NIS+ uses itself to store information that it exports.

The zo\_owner member contains a fully qualified NIS+ name of the form *principal.domain*. This name is called a NIS+ principal name and is used to identify authentication information in a credential table. When the server constructs a search query of the form

[cname=principal], cred.org\_dir.domain.

The query will return to the server credential information about *principal* for all flavors of RPC authentication that are in use by that principal. When an RPC request is made to the server, the authentication flavor is extracted from the request and is used to find out the NIS+ principal name of the client. For example, if the client is using the AUTH\_DES authentication flavor, it will include in the authentication credentials the network name or *netname* of the user making the request. This netname will be of the form

unix.UID@domain

The NIS+ server will then construct a query on the credential database of the form

[auth\_name=netname,auth\_type=AUTH\_DES],cred.org\_dir.domain.

This query will return an entry which contains a principal name in the first column. This NIS+ principal name is used to control access to NIS+ objects.

The group owner for the object is treated differently. The group owner member is optional (it should be the null string if not present) but must be fully qualified if present. A group name takes the form

group.domain.

which the server then maps into a name of the form

group.groups\_dir.domain.

The purpose of this mapping is to prevent NIS+ group names from conflicting with user specified domain or table names. For example, if a domain was called *engineering.foo.com.*, then without the mapping a NIS+ group of the same name to represent members of engineering would not be possible. The contents of groups are lists of NIS+ principal names which are used exactly like the zo\_owner name in the object. See nis groups(3NSL) for more details.

The zo\_access member contains the bitmask of access rights assigned to this object. There are four access rights defined, and four are reserved for future use and must be zero. This group of 8 access rights can be granted to four categories of client. These categories are the object's owner, the object's group owner, all authenticated clients (world), and all unauthenticated clients (nobody). Note that access granted to "nobody" is really access granted to everyone, authenticated and unauthenticated clients.

The zo\_ttl member contains the number of seconds that the object can "live" in a cache before it is expired. This value is called the time to live for this object. This number is particularly important on group and directory (domain) objects. When an

object is cached, the current time is added to the value in zo\_ttl. Then each time the cached object is used, the time in zo\_ttl is compared with the current time. If the current time is later than the time in zo\_ttl the object is said to have expired and the cached copy should not be used.

Setting the TTL is somewhat of an art. You can think of it as the "half life" of the object, or half the amount of time you believe will pass before the object changes. The benefit of setting the ttl to a large number is that the object will stay in a cache for long periods of time. The problem with setting it to a large value is that when the object changes it will take a long time for the caches to flush out old copies of that object. The problems and benefits are reversed for setting the time to a small value. Generally setting the value to 43200 (12 hrs) is reasonable for things that change day to day, and 3024000 is good for things that change week to week. Setting the value to 0 will prevent the object from ever being cached since it would expire immediately.

The zo\_data member is a discriminated union with the following members:

```
zotypes zo_type;
union {
    struct directory_obj di_data;
    struct group_obj gr_data;
    struct table_obj ta_data;
    struct entry_obj en_data;
    struct link_obj li_data;
    struct {
        uint_t po_data_len;
        char *po_data_val;
    } po_data;
    } objdata u;
```

The union is discriminated based on the type value contained in zo\_type. There six types of objects currently defined in the NIS+ service. These types are the directory, link, group, table, entry, and private types.

```
enum zotypes {
    BOGUS_OBJ = 0,
    NO_OBJ = 1,
    DIRECTORY_OBJ = 2,
    GROUP_OBJ = 3,
    TABLE_OBJ = 4,
    ENTRY_OBJ = 5,
    LINK_OBJ = 6,
    PRIVATE_OBJ = 7
 };
 typedef enum zotypes zotypes;
```

All object types define a structure that contains data specific to that type of object. The simplest are private objects which are defined to contain a variable length array of octets. Only the owner of the object is expected to understand the contents of a private object. The following section describe the other five object types in more significant detail.

**Directory Objects** The first type of object is the *directory* object. This object's variant part is defined as follows: enum nstype { UNKNOWN = 0, NIS = 1, SUNYP = 2, DNS = 4, = 5, X500 DNANS = 6, XCHS = 7, } typedef enum nstype nstype; struct oar\_mask { uint t oa rights; zotypes oa\_otype; } typedef struct oar mask oar mask; struct endpoint { char \*uaddr; char \*family; char \*proto; } typedef struct endpoint endpoint; struct nis server { nis name name; struct { uint\_t ep\_len; endpoint \*ep\_val; } ep; uint\_t key\_type; netobj pkey; typedef struct nis server nis server; struct directory\_obj { nis\_name do\_name; nstype do\_type; struct { uint\_t do\_servers\_len; nis server \*do servers val; } do\_servers; uint32 t do ttl; struct { uint\_t do\_armask\_len; oar mask \*do armask val; } do\_armask; } typedef struct directory\_obj directory\_obj;

The main structure contains five primary members: do\_name, do\_type, do\_servers, do\_ttl, and do\_armask. The information in the do\_servers structure is sufficient for the client library to create a network connection with the named server for the directory.

The do\_name member contains the name of the directory or domain represented in a format that is understandable by the type of nameservice serving that domain. In the

case of NIS+ domains, this is the same as the name that can be composed using the zo\_name and zo\_domain members. For other name services, this name will be a name that they understand. For example, if this were a directory object describing an X.500 namespace that is "under" the NIS+ directory *eng.sun.com.*, this name might contain "/C=US, /O=Sun Microsystems, /OU=Engineering/". The type of nameservice that is being described is determined by the value of the member do\_type.

The do\_servers structure contains two members. do\_servers\_val is an array of *nis\_server* structures; do\_servers\_len is the number of cells in the array. The *nis\_server* structure is designed to contain enough information such that machines on the network providing name services can be contacted without having to use a name service. In the case of NIS+ servers, this information is the name of the machine in *name*, its public key for authentication in *pkey*, and a variable length array of endpoints, each of which describes the network endpoint for the rpcbind daemon on the named machine. The client library uses the addresses to contact the server using a transport that both the client and server can communicate on and then queries the rpcbind daemon to get the actual transport address that the server is using.

Note that the first server in the *do\_servers* list is always the master server for the directory.

The *key\_type* field describes the type of key stored in the *pkey* netobj (see /usr/include/rpc/xdr.h for a definition of the network object structure). Currently supported types are NIS\_PK\_NONE for no public key, NIS\_PK\_DH for a Diffie-Hellman type public key, and NIS\_PK\_DHEXT for an extended Diffie-Hellman public key.

The do\_ttl member contains a copy of the zo\_ttl member from the common attributes. This is the duplicated because the cache manager only caches the variant part of the directory object.

The do\_armask structure contains two members. do\_armask\_val is an array of oar\_mask structures; do\_armask\_len is the number of cells in the array. The oar\_mask structure contains two members: oa\_rights specifies the access rights allowed for objects of type oa\_otype. These access rights are used for objects of the given type in the directory when they are present in this array.

The granting of access rights for objects contained within a directory is actually two-tiered. If the directory object itself grants a given access right (using the zo\_access member in the nis\_object structure representing the directory), then all objects within the directory are allowed that access. Otherwise, the do\_armask structure is examined to see if the access is allowed specifically for that type of structure. This allows the administrator of a namespace to set separate policies for different object types, for example, one policy for the creation of tables and another policy for the creation of other directories. See nis+(1) for more details.

Link Objects

Link objects provide a means of providing *aliases* or symbolic links within the namespace. Their variant part is defined as follows.

```
struct link_obj {
    zotypes li_rtype;
    struct {
        uint_t li_attrs_len;
        nis_attr *li_attrs_val;
    } li_attrs;
    nis_name li_name;
}
```

The li\_rtype member contains the object type of the object pointed to by the link. This is only a hint, since the object which the link points to may have changed or been removed. The fully qualified name of the object (table or otherwise) is specified in the member li\_name.

NIS+ links can point to either other objects within the NIS+ namespace, or to entries within a NIS+ table. If the object pointed to by the link is a table and the member li\_attrs has a nonzero number of attributes (index name/value pairs) specified, the table is searched when this link is followed. All entries which match the specified search pattern are returned. Note, that unless the flag FOLLOW\_LINKS is specified, the nis lookup(3NSL) function will always return non-entry objects.

**Group Objects** Group objects contain a membership list of NIS+ principals. The group objects' variant part is defined as follows.

```
struct group_obj {
    uint_t gr_flags;
    struct {
        uint_t gr_members_len;
        nis_name *gr_members_val;
    } gr_members;
}
```

The gr\_flags member contains flags that are currently unused. The gr\_members structure contains the list of principals. For a complete description of how group objects are manipulated see nis groups(3NSL).

**Table Objects**The NIS+ table object is analogous to a YP map. The differences stem from the access<br/>controls, and the variable schemas that NIS+ allows. The table objects data structure is<br/>defined as follows:

```
#define TA_BINARY 1
#define TA_CRYPT 2
#define TA_XDR 4
#define TA_SEARCHABLE 8
#define TA_CASE 16
#define TA_MODIFIED 32
struct table_col {
    char *tc_name;
    uint_t tc_flags;
    uint_t tc_rights;
}
```

388 man pages section 3: Networking Library Functions • Last Revised 10 Nov 1999

```
typedef struct table_col table_col;
struct table_obj {
    char *ta_type;
    uint_t ta_maxcol;
    uchar_t ta_sep;
    struct {
        uint_t ta_cols_len;
        table_col *ta_cols_val;
    } ta_cols;
    char *ta_path;
}
```

The ta\_type member contains a string that identifies the type of entries in this table. NIS+ does not enforce any policies as to the contents of this string. However, when entries are added to the table, the NIS+ service will check to see that they have the same "type" as the table as specified by this member.

The structure ta\_cols contains two members. ta\_cols\_val is an array of table\_col structures. The length of the array depends on the number of columns in the table; it is defined when the table is created and is stored in ta\_cols\_len. ta\_maxcol also contains the number of columns in the table and always has the same value as ta cols len. Once the table is created, this length field cannot be changed.

The ta\_sep character is used by client applications that wish to print out an entry from the table. Typically this is either space (" ") or colon (":").

The ta\_path string defines a concatenation path for tables. This string contains an ordered list of fully qualified table names, separated by colons, that are to be searched if a search on this table fails to match any entries. This path is only used with the flag FOLLOW\_PATH with a nis\_list() call. See nis\_tables(3NSL) for information on these flags.

In addition to checking the type, the service will check that the number of columns in an entry is the same as those in the table before allowing that entry to be added.

Each column has associated with it a name in tc\_name, a set of flags in tc\_flags, and a set of access rights in tc\_rights. The name should be indicative of the contents of that column.

The TA\_BINARY flag indicates that data in the column is binary (rather than text). Columns that are searchable cannot contain binary data. The TA\_CRYPT flag specifies that the information in this column should be encrypted prior to sending it over the network. This flag has no effect in the export version of NIS+. The TA\_XDR flag is used to tell the client application that the data in this column is encoded using the XDR protocol. The TA\_BINARY flag must be specified with the XDR flag. Further, by convention, the name of a column that has the TA\_XDR flag set is the name of the XDR function that will decode the data in that column.

The TA\_SEARCHABLE flag specifies that values in this column can be searched. Searchable columns must contain textual data and must have a name associated with them. The flag TA\_CASE specifies that searches involving this column ignore the case

of the value in the column. At least one of the columns in the table should be searchable. Also, the combination of all searchable column values should uniquely select an entry within the table. The TA\_MODIFIED flag is set only when the table column is modified. When TA\_MODIFIED is set, and the object is modified again, the modified access rights for the table column must be copied, not the default access rights.

**Entry Objects** Entry objects are stored in tables. The structure used to define the entry data is as follows.

```
#define EN BINARY
                  1
#define EN CRYPT 2
#define EN XDR 4
#define EN_MODIFIED
struct entry_col {
   uint t ec flags;
   struct {
      uint_t ec_value_len;
char *ec_value_val;
   } ec value;
}
typedef struct entry_col entry_col;
struct entry_obj {
   char *en_type;
   struct {
      uint_t en_cols_len;
      entry_col *en_cols_val;
   } en cols;
}
```

The en\_type member contains a string that specifies the type of data this entry represents. The NIS+ server will compare this string to the type string specified in the table object and disallow any updates or modifications if they differ.

The en\_cols structure contains two members: en\_cols\_len and en\_cols\_val. en\_cols\_val is an array of entry\_col structures. en\_cols\_len contains a count of the number of cells in the en\_cols\_val array and reflects the number of columns in the table -- it always contains the same value as the table\_obj.ta\_cols.ta\_cols\_len member from the table which contains the entry.

The entry\_col structure contains information about the entry's per-column values. ec\_value contains information about a particular value. It has two members: ec\_value\_val, which is the value itself, and ec\_value\_len, which is the length (in bytes) of the value. entry\_col also contains the member ec\_flags, which contains a set of flags for the entry.

The flags in ec\_flags are primarily used when adding or modifying entries in a table. All columns that have the flag EN\_CRYPT set will be encrypted prior to sending them over the network. Columns with EN\_BINARY set are presumed to contain binary data. The server will ensure that the column in the table object specifies binary data prior to allowing the entry to be added. When modifying entries in a table, only those

	columns that have changed need be sent to the server. Those columns should each have the EN_MODIFIED flag set to indicate this to the server.
SEE ALSO	<pre>nis+(1), nis_groups(3NSL), nis_names(3NSL), nis_server(3NSL), nis_subr(3NSL), nis_tables(3NSL)</pre>

nis\_ping(3NSL)

NAME	nis_ping, nis_checkpoint – misc NIS+ log administration functions	
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>	
	<pre>void nis_ping(nis_name dirname, ui</pre>	<pre>nt32_t utime, nis_object *dirobj);</pre>
	<pre>nis_result *nis_checkpoint(nis_n</pre>	ame <i>dirname</i> );
DESCRIPTION	nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter dirname identifies the directory with the change. If the parameter <i>dirobj</i> is NULL, this function looks up the directory obje for dirname and uses the list of replicas it contains. The parameter <i>utime</i> contains th timestamp of the last change made to the directory. This timestamp is used by the replicas when retrieving updates made to the directory.	
	The effect of calling nis_ping() is to schedule an update on the replica. A short time after a ping is received, typically about two minutes, the replica compares the last update time for its databases to the timestamp sent by the ping. If the ping timestamp is later, the replica establishes a connection with the master server and request all changes from the log that occurred after the last update that it had recorded in its local log.	
	<pre>nis_checkpoint() is used to force the service to checkpoint information that has been entered in the log but has not been checkpointed to disk. When called, this function checkpoints the database for each table in the directory, the database containing the directory and the transaction log. Care should be used in calling this function since directories that have seen a lot of changes may take several minutes to checkpoint. During the checkpointing process, the service will be unavailable for updates for all directories that are served by this machine as master.</pre>	
	<pre>nis_checkpoint() returns a pointer to a <i>nis_result</i> structure (described in nis_tables(3NSL)). This structure should be freed with nis_freeresult() (see nis_names(3NSL)). The only items of interest in the returned result are the status value and the statistics.</pre>	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	nislog(1M), nis_names(3NSL), nis_tak attributes(5)	oles(3NSL), nisfiles(4),

392 man pages section 3: Networking Library Functions • Last Revised 17 Feb 1998

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>
	<pre>nis_error nis_mkdir(nis_name dirname, nis_server *machine);</pre>
	<pre>nis_error nis_rmdir(nis_name dirname, nis_server *machine);</pre>
	<pre>nis_error nis_servstate(nis_server *machine, nis_tag *tags, int</pre>
	<pre>nis_error nis_stats(nis_server *machine, nis_tag *tags, int numtags,</pre>
	<pre>void nis_freetags(nis_tag *tags, int numtags);</pre>
	<pre>nis_server **nis_getservlist(nis_name dirname);</pre>
	<pre>void nis_freeservlist(nis_server **machines);</pre>
DESCRIPTION	These functions provide a variety of services for NIS+ applications.
	<pre>nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, dirname, on a server, machine. If this operation is successful, it means that the directory object describing dirname has been updated to reflect that server machine is serving the named directory. For a description of the nis_server structure, refer to nis_objects(3NSL).</pre>
	Per-server and per-directory access restrictions may apply to nis_mkdir(). See nisopaccess(1)
	<pre>nis_rmdir() is used to delete the directory, dirname, from the specified server machine. The machine parameter cannot be NULL. Note that nis_rmdir() does not remove the directory dirname 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 nis_remove() to remove the directory dirname from the namespace and call nis_rmdir() 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 nis_modify() to remove the server from the directory object and then call nis_rmdir() to remove the replica.</pre>
	Per-server and per-directory access restrictions may apply to nis_rmdir(). See nisopaccess(1)
	For a description of the nis_server structure, refer to nis_objects(3NSL).
	nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried.

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\_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 refer to nis\_objects(3NSL). nis\_freeservlist() frees the list of servers 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

SEE ALSO nisopaccess(1), nisstat(1M), nis\_names(3NSL), nis\_objects(3NSL), nis\_subr(3NSL), attributes(5)

NAME	<pre>nis_subr, nis_leaf_of, nis_name_of, nis_domain_of, nis_getnames, nis_freenames, nis_dir_cmp, nis_clone_object, nis_destroy_object, nis_print_object - NIS+ subroutines</pre>		
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpcsvc nis.h=""></rpcsvc></pre>		
	<pre>nis_name nis_leaf_of(const nis_name name);</pre>		
	<pre>nis_name nis_name_of(const nis_name name);</pre>		
	<pre>nis_name nis_domain_of(const nis_name name);</pre>		
	<pre>nis_name *nis_getnames(const nis_name name);</pre>		
	<pre>void nis_freenames(nis_name *namelist);</pre>		
	<pre>name_pos nis_dir_cmp(const nis_name n1, const nis_name n2);</pre>		
	<pre>nis_object *nis_clone_object(const nis_object *src, nis_object</pre>		
	<pre>void nis_destroy_object(nis_object *obj);</pre>		
	<pre>void nis_print_object(const nis_object *obj);</pre>		
DESCRIPTION	These subroutines are provided to assist in the development of NIS+ applications. They provide several useful operations on both NIS+ names and objects.		
	The first group, nis_leaf_of(), nis_domain_of(), and nis_name_of() provid the functions for parsing NIS+ names. nis_leaf_of() will return the first label in a NIS+ name. It takes into account the double quote character "" which can be used to protect embedded '.' (dot) characters in object names. Note that the name returned will never have a trailing dot character. If passed the global root directory name ".", it will return the null string.		
	<pre>nis_domain_of() returns the name of the NIS+ domain in which an object resides. This name will always be a fully qualified NIS+ name and ends with a dot. By iteratively calling nis_leaf_of() and nis_domain_of() it is possible to break a NIS+ name into its individual components.</pre>		
	<pre>nis_name_of() is used to extract the unique part of a NIS+ name. This function removes from the tail portion of the name all labels that are in common with the local domain. Thus if a machine were in domain foo.bar.baz. and nis_name_of() were passed a name bob.friends.foo.bar.baz, then nis_name_of() would return the unique part, bob.friends. If the name passed to this function is not in either the local domain or one of its children, this function will return null.</pre>		
	<pre>nis_getnames() will return a list of candidate names for the name passed in as name. If this name is not fully qualified, nis_getnames() will generate a list of names using the default NIS+ directory search path, or the environment variable NIS_PATH if it is set. The returned array of pointers is terminated by a NULL pointer, and the memory associated with this array should be freed by calling nis_freenames().</pre>		

# nis\_subr(3NSL)

SEE ALSO	nis_names(3NSI	L),nis_objects(3NSL),	nis_tables(3NSL),attributes(5)	
	MT-Level		Safe	
	ATTRIBUTE TYPE		ATTRIBUTE VALUE	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
ENVIRONMENT VARIABLES	NIS_PATH	put. Its primary use is for debugging NIS+ programs. This variable overrides the default NIS+ directory search path used by nis_getnames(). It contains an ordered list of directories separated by ':' (colon) characters. The '\$' (dollar sign) character is treated specially. Directory names that end in '\$' have the default domain appended to them, and a '\$' by itself is replaced by the list of directories between the default domain and the global root that are at least two levels deep. The default NIS+ directory search path is '\$'.		
	<pre>nis_destroy_object() can be used to destroy an object created by nis_clone_object(). This will free up all memory associated with the object and free the pointer passed. If the object was cloned into an array (using the <i>dest</i> parameter to nis_clone_object()) then the object <i>cannot</i> be freed with this function. Instead, the function xdr_free(xdr_nis_object, <i>dest</i>) must be used. nis_print_object() prints out the contents of a NIS+ object structure on the</pre>			
	The second set of functions, consisting of nis_clone_object() and nis_destroy_object(), are used for manipulating objects. nis_clone_object() creates an exact duplicate of the NIS+ object <i>src</i> . If the value of <i>dest</i> is non-null, it creates the clone of the object into this object structure and allocate the necessary memory for the variable length arrays. If this parameter is null, a pointer to the cloned object is returned. Refer to nis_objects(3NSL) for a description of the nis_object structure.			
	Though nis_dir_cmp() can be used to compare any two NIS+ names, it is used primarily to compare domain names. This comparison is done in a case independent fashion, and the results are an enum of type name_pos. When the names passed to this function are identical, the function returns a value of SAME_NAME. If the name <i>n1</i> is a direct ancestor of name <i>n2</i> , then this function returns the result HIGHER_NAME. Similarly, if the name <i>n1</i> is a direct descendant of name <i>n2</i> , then this function returns the result ancestor nor a direct descendant of <i>n2</i> , as it would be if the two names were siblings in separate portions of the namespace, then this function returns the result NOT_SEQUENTIAL. Finally, if either name cannot be parsed as a legitimate name then this function returns the value BAD_NAME.			

nis\_subr(3NSL)

**NOTES** | nis\_leaf\_of(), nis\_name\_of() and nis\_clone\_object() return their results as thread-specific data in multithreaded applications.

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

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a "soft" success or a "soft" failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.	
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).	
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <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 nis_getnames() (see nis_local_names(3NSL)) which uses the environment variable NIS_PATH.	
RETURN_RESULT	This flag is used to specify that a copy of the returning object be returned in the nis_result structure if the operation was successful.	
The third parameter to nis_list(), <i>callback</i> , 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 <i>nis_result</i> 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.

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.

nis\_modify\_entry() modifies an object identified by *name*. The parameter *object* should point to an entry with the EN\_MODIFIED flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object*: zo\_owner, zo\_group, and zo\_access.

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

If the flags parameter contains the flag MOD\_SAMEOBJ then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails

	with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.	
	If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful.	
	<pre>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 <i>name</i>. If a search criteria is present in <i>name</i> it is ignored. The value of <i>cookie</i> within the nis_result structure must be copied by the caller into local storage and passed as an argument to nis_next_entry().</pre>	
	<pre>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.</pre>	
<b>RETURN VALUES</b>	These functions return a pointer to a structure of type nis_result:	
	<pre>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)).</pre>	
	The objects structure contains two members. <i>objects_val</i> is an array of <i>nis_object</i> structures; <i>objects_len</i> 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.	

0(01(02)				
	dticks	time a database cal	the database backend, this time is measured from the ll starts, until a result is returned. If the request results in e database, this is the sum of all the time spent in those	
	<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.			
	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 o the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.		
	service co in <i>cticks</i> w	cting the value in <i>dticks</i> from the value in <i>zticks</i> will yield the time spent in the code itself. Subtracting the sum of the values in <i>zticks</i> and <i>aticks</i> from the value s will yield the time spent in the client library itself. Note: all of the tick times asured in microseconds.		
ERRORS		library can return a es are documented b	variety of error returns and diagnostics. The more pelow.	
	NIS_BADA	ATTRIBUTE	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_BADI	NAME	The name passed to the function is not a legal NIS+ name.	
	NIS_BADI	REQUEST	A problem was detected in the request structure passed to the client library.	
	NIS_CAC	HEEXPIRED	The entry returned came from an object cache that has <i>expired</i> . This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.	
	NIS_CBEI	RROR	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_CBRI	ESULTS	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_FOR	EIGNNS	The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type	

	DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.
NIS_INVALIDOBJ	The object pointed to by <i>object</i> is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.
NIS_LINKNAMEERROR	The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.
NIS_MODFAIL	The attempted modification failed for some reason.
NIS_NAMEEXISTS	An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.
NIS_NAMEUNREACHABLE	This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.
NIS_NOCALLBACK	The server was unable to contact the callback service on your machine. This results in no data being returned.
NIS_NOMEMORY	Generally a fatal result. It means that the service ran out of heap space.
NIS_NOSUCHNAME	This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.
NIS_NOSUCHTABLE	The named table does not exist.
NIS_NOT_ME	A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.
NIS_NOTFOUND	No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the nis_remove().
I	

	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

		server is cur returned to r	<pre>try() return this error when the master rently updating its internal state. It can be nis_list() when the function specifies a the server does not have the resources to acks.</pre>
	NIS_TYPEMISMATCH		was made to add or modify an entry in a e entry passed was of a different type than
ENVIRONMENT VARIABLES		t, this variable i	is the search path used by nis_list() if is set.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		following attributes:
	ATTRIBUTE TYPE	E	ATTRIBUTE VALUE
	MT-Level		MT-Safe with exceptions
SEE ALSO	<pre>niscat(1), niserror(1), nismatch(1), nis_cachemgr(1M), nis_clone_object(3NSL), n, nis_destroy_object(3NSL), nis_error(3NSL), nis_getnames(3NSL), nis_local_names(3NSL), nis_names(3NSL), nis_objects(3NSL), nis_server(3NSL), rpc_svc_calls(3NSL), syslog(3C), attributes(5)</pre>		
WARNINGS	Use the flag HARD_LOOKUP carefully since it can cause the application to block indefinitely during a network partition.		t can cause the application to block
NOTES	The path used when the flag FOLLOW_PATH is specified, is the one present in the <i>first</i> 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.		
	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		

nlsgetcall(3NSL)

NAME	nlsgetcall – get client's data passed via the listener		
SYNOPSIS	<pre>#include <sys tiuser.h=""></sys></pre>		
	<pre>struct t_call *nlsgetcall(int fildes); nlsgetcall() allows server processes started by the listener process to access the client's t_call structure, that is, the <i>sndcall</i> argument of t_connect(3NSL).</pre>		
DESCRIPTION			
	The t_call structure returned by nlsgetcall() can be released using t_free( $3NSL$ ).		
	nlsgetcall() returns the address of an allocated t_call structure or NULL if a t_call structure cannot be allocated. If the t_alloc() succeeds, undefined environment variables are indicated by a negative <i>len</i> field in the appropriate netbuf structure. A <i>len</i> field of zero in the netbuf structure is valid and means that the original buffer in the listener's t_call structure was NULL.		
RETURN VALUES	A NULL pointer is returned if a t_call structure cannot be allocated by t_alloc(). t_errno can be inspected for further error information. Undefined environment variables are indicated by a negative length field ( <i>len</i> ) in the appropriate netbuf structure.		
FILES	/usr/lib/libnsl_s.a		
	/usr/lib/libslan.a		
	/usr/lib/libnls.a		
	/usr/lib/libnls.a		
ATTRIBUTES	<pre>/usr/lib/libnls.a See attributes(5) for descriptions of the</pre>	following attributes:	
ATTRIBUTES		following attributes:	
ATTRIBUTES		following attributes:	
ATTRIBUTES	See attributes(5) for descriptions of the		
ATTRIBUTES SEE ALSO	See attributes(5) for descriptions of the           ATTRIBUTE TYPE	ATTRIBUTE VALUE Unsafe SL), t_connect(3NSL), t_error(3NSL),	
	See attributes(5) for descriptions of the           ATTRIBUTE TYPE           MT-Level           nlsadmin(1M), getenv(3C), t_alloc(3N)	ATTRIBUTE VALUE         Unsafe         SL), t_connect(3NSL), t_error(3NSL), es(5)         ned as being unsigned. In order to check	
SEE ALSO	See attributes(5) for descriptions of the ATTRIBUTE TYPE MT-Level nlsadmin(1M), getenv(3C), t_alloc(3N t_free(3NSL), t_sync(3NSL), attribut The <i>len</i> field in the netbuf structure is defi for error returns, it should first be cast to an	ATTRIBUTE VALUE         Unsafe         SL), t_connect(3NSL), t_error(3NSL), es(5)         ned as being unsigned. In order to check in int.         wer data ( <i>udata</i> ) and options data ( <i>opt</i> ) to 128	
SEE ALSO	See attributes(5) for descriptions of the ATTRIBUTE TYPE MT-Level nlsadmin(1M), getenv(3C), t_alloc(3N t_free(3NSL), t_sync(3NSL), attribut The <i>len</i> field in the netbuf structure is defi for error returns, it should first be cast to ar The listener process limits the amount of us bytes each. Address data <i>addr</i> is limited to 6	ATTRIBUTE VALUE         Unsafe         SL), t_connect(3NSL), t_error(3NSL), es(5)         ned as being unsigned. In order to check nint.         eer data (udata) and options data (opt) to 128 54 bytes. If the original data was longer, no	
SEE ALSO WARNINGS	See attributes(5) for descriptions of the ATTRIBUTE TYPE MT-Level nlsadmin(1M), getenv(3C), t_alloc(3N t_free(3NSL), t_sync(3NSL), attribut The <i>len</i> field in the netbuf structure is defi for error returns, it should first be cast to an The listener process limits the amount of us bytes each. Address data <i>addr</i> is limited to 6 indication of overflow is given.	ATTRIBUTE VALUE         Unsafe         SL), t_connect(3NSL), t_error(3NSL), es(5)         ned as being unsigned. In order to check nint.         ter data ( <i>udata</i> ) and options data ( <i>opt</i> ) to 128 54 bytes. If the original data was longer, no         before calling this routine.	

### nlsprovider(3NSL)

NAME	nlsprovider – get name of transport provider		
SYNOPSIS	char * <b>nlsprovider</b> (void);		
DESCRIPTION	nlsprovider() returns a pointer to a null-terminated character string which contains the name of the transport provider as placed in the environment by the listener process. If the variable is not defined in the environment, a NULL pointer is returned.		
	The environment variable is only available to server processes started by the listener process.		
<b>RETURN VALUES</b>	If the variable is not defined in the environment, a NULL pointer is returned.		
FILES	/usr/lib/libslan.a (7300)		
	/usr/lib/libnls.a (3B2 Computer)		
	/usr/lib/libnsl_s.a		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

### **SEE ALSO** | nlsadmin(1M), attributes(5)

**NOTES** This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

nlsrequest(3NSL)
------------------

NAME	nlsrequest – format and send listener service request message		
SYNOPSIS	<pre>#include <listen.h></listen.h></pre>		
	<pre>int nlsrequest(int fildes, char *service_code);</pre>		
	<pre>extern int _nlslogt_errno; extern char *_nlsrmsg;</pre>		
DESCRIPTION	Given a virtual circuit to a listener process ( <i>fildes</i> ) and a service code of a server process, nlsrequest() formats and sends a <i>service request message</i> to the remote listener process requesting that it start the given service. nlsrequest() waits for the remote listener process to return a <i>service request response message</i> , which is made available to the caller in the static, null-terminated data buffer pointed to by _nlsrmsg. The <i>service request response message</i> includes a success or failure code and a text message. The entire message is printable.		
RETURN VALUES	The success or failure code is the integer return code from nlsrequest(). Zero indicates success, other negative values indicate nlsrequest() failures as follows:		
	-1 Error encountered by nlsreque	est(), see t_errno.	
	Positive values are error return codes from the <i>listener</i> process. Mnemonics for these codes are defined in <listen.h>.</listen.h>		
	2 Request message not interpretab	ble.	
	3 Request service code unknown.		
	4 Service code known, but currently disabled.		
	If non-null, _nlsrmsg contains a pointer to a static, null-terminated character buffer containing the <i>service request response message</i> . Note that both _nlsrmsg and the data buffer are overwritten by each call to nlsrequest().		
	If _nlslog is non-zero, nlsrequest() prints error messages on stderr. Initially, _nlslog is zero.		
FILES	/usr/lib/libnls.a		
	/usr/lib/libslan.a		
	/usr/lib/libnsl_s.a		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level	Unsafe	
		J	
SEE ALSO	<pre>nlsadmin(1M), t_error(3NSL), t_snd(3</pre>	NSL),t_rcv(3NSL),attributes(5)	

- WARNINGS | nlsrequest() cannot always be certain that the remote server process has been successfully started. In this case, nlsrequest() returns with no indication of an error and the caller will receive notification of a disconnect event by way of a T\_LOOK error before or during the first t\_snd() or t\_rcv() call.
  - **NOTES** These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

rcmd(3SOCKET)	
NAME	rcmd, rcmd_af, rresvport, rresvport_af, ruserok – routines for returning a stream to a remote command
SYNOPSIS	cc [ flag ] filelsocket -lnsl [ library ]
	<pre>int rcmd(char **ahost, unsigned short inport, const char *luser, const char *ruser, const char *cmd, int *fd2p);</pre>
	<pre>int rcmd_af(char **ahost, unsigned short inport, const char *luser, const char *ruser, const char *cmd, int *fd2p, int af);</pre>
	<pre>int rresvport(int *port);</pre>
	<pre>int rresvport_af(int *port, int af);</pre>
	<pre>int ruserok(const char *rhost, int suser, const char *ruser, const char *luser);</pre>
DESCRIPTION	<pre>rcmd() is a routine used by the superuser to execute a command on a remote machine using an authentication scheme based on reserved port numbers. It is assumed that an AF_INET socket is returned with rcmd().rcmd_af() allows the application to choose which type of socket is returned by passing in the address family, either AF_INET or AF_INET6.</pre>
	<pre>rresvport() is a routine that returns a descriptor to a socket with an address in the privileged port space. rresvport_af() is equivalent to rresvport(), except that you can choose the type of socket address family that will be returned by rresvport_af(), either AF_INET or AF_INET6.</pre>
	$\tt ruserok()$ is a routine used by servers to authenticate clients requesting service with $\tt rcmd.$
	All of these functions are present in the same file and are used by the in.rshd(1M) server (among others).
	<pre>rcmd() and rcmd_af() look up the host *ahost using getipnodebyname(3SOCKET), returning -1 if the host does not exist. Otherwise *ahost is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port inport.</pre>
	If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller, and given to the remote command as its standard input (file descriptor 0) and standard output (file descriptor 1). If $fd2p$ is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in $*fd2p$ . The control process will return diagnostic output from the command (file descriptor 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If $fd2p$ is 0, then the standard error (file descriptor 2) of the remote command will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

	The protocol is described in detail in in.rs	shd(1M).		
	The rresvport() and rresvport_af() routines are used to obtain a socket bound to a privileged port number. This socket is suitable for use by rcmd() and rresvport_af() and several other routines. Privileged Internet ports are those in the range 1 to 1023. Only the superuser is allowed to bind a socket to a privileged port number. The application must pass in <i>port</i> , which must be in the range 512 to 1023. The system first tries to bind to that port number. If it fails, the system then tries to bind to another unused privileged port, if one is available.			
	<pre>ruserok() takes a remote host's name, as returned by a gethostbyaddr() routine, two user names and a flag indicating whether the local user's name is that of the superuser. See gethostbyname(3NSL). It then checks the files /etc/hosts.equiv and possibly .rhosts in the local user's home directory to see if the request for service is allowed. 0 is returned if the machine name is listed in the /etc/hosts.equiv file, or the host and remote user name are found in the .rhosts file; otherwise ruserok() returns -1. If the superuser flag is 1, the checking of the /etc/hosts.equiv file is bypassed.</pre>			
RETURN VALUES	<pre>rcmd() and rcmd_af() return a valid soc -1 upon error and print a diagnostic messa</pre>			
	<pre>rresvport() and rresvport_af() retu success. They return -1 upon error with the reason for failure.</pre>			
FILES	/etc/hosts.equiv system trust	ed hosts and users		
	~/.rhosts user's trusted hosts and users			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ATTRIBUTE TYPE	ATTRIBUTE VALUE		
	MT-Level	Unsafe		
SEE ALSO	rlogin(1), rsh(1), in.rexecd(1M), in.r gethostbyname(3NSL), getipnodebyna attributes(5)			
NOTES	The error code EAGAIN is overloaded to mean "All network ports in use."			
	These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.			

# recv(3SOCKET)

CV(SSOCKET)					
NAME	recv, recvfrom, recvr	nsg – receive a n	message from a socket		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""> #include <sys uio.h=""></sys></sys></sys></pre>				
	ssize_t <b>recv</b> (int	t $s$ , void * $buf$	f, size_t len, int flags);		
		<b>m</b> (int <i>s</i> , void <i>com</i> , int * <i>from</i>	d *buf, size_t len, int flags, struct llen);		
	ssize_t recvmsg	(int s, struc	ct msghdr * <i>msg</i> , int <i>flags</i> );		
DESCRIPTION	<pre>socket.recv() may while recvfrom()</pre>	be used only or and recymsg()	<ul> <li>are used to receive messages from another</li> <li>n a <i>connected</i> socket (see connect(3SOCKET)),</li> <li>) may be used to receive data on a socket whether</li> <li>a socket created with socket(3SOCKET).</li> </ul>		
	If <i>from</i> is not a NULL pointer, the source address of the message is filled in. <i>from</i> value-result parameter, initialized to the size of the buffer associated with <i>from</i> modified on return to indicate the actual size of the address stored there. The the message is returned. If a message is too long to fit in the supplied buffer, of bytes may be discarded depending on the type of socket the message is received (see socket(3SOCKET)).				
	If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see fcntl(2)) in which case -1 is returned with the external variable errno set to EWOULDBLOCK.				
	The select() call may be used to determine when more data arrives.				
	The <i>flags</i> parameter is formed by ORing one or more of the following:				
	r				
	The recvmsg() call uses a msghdr structure to minimize the number of directly supplied parameters. This structure is defined in <sys socket.h=""> and includes t following members:</sys>				
	int msg struct iovec *ms int msg caddr_t msg	g_iov;	<pre>/* optional address */ /* size of address */ /* scatter/gather array */ /* # elements in msg_iov */ /* access rights sent/received */</pre>		

	Here msg_name and msg_namelen specify the destination address if the socket is unconnected; msg_name may be given as a NULL pointer if no names are desired or required. The msg_iov and msg_iovlen describe the scatter-gather locations, as described in read(2). A buffer to receive any access rights sent along with the message is specified in msg_accrights, which has length msg_accrightslen.			
<b>RETURN VALUES</b>	These calls return the number	of bytes recei	ived, or $-1$ if an error occurred.	
ERRORS	The calls fail if:			
	EBADF	<i>s</i> is an invali	id file descriptor.	
	EINTR	-	on was interrupted by delivery of a signal lata was available to be received.	
	EIO	An I/O erro the file syste	r occurred while reading from or writing to m.	
	ENOMEM	There was ir operation to	nsufficient user memory available for the complete.	
	ENOSR		insufficient STREAMS resources available ation to complete.	
	ENOTSOCK	s is not a soc	ket.	
	ESTALE	A stale NFS file handle exists.		
	EWOULDBLOCK	The socket is operation w	s marked non-blocking and the requested ould block.	
ATTRIBUTES	See attributes(5) for descr	iptions of the	following attributes:	
	ATTRIBUTE TYPE		ATTRIBUTE VALUE	
	MT-Level		Safe	
SEE ALSO	fcntl(2), ioctl(2), read(2),	,	OCKET), getsockopt(3SOCKET), ributes(5), socket(3HEAD)	

### recv(3XNET)

NAME	recv – receive a message from a connected socket			
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>			
	ssize_t <b>recv</b> (i	int <i>socket</i> , void * <i>buffer</i> , siz	<pre>ze_t length, int flags);</pre>	
DESCRIPTION	The recv() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data. The function takes the following arguments:			
	socket	Specifies the socket file descri	iptor.	
	buffer	Points to a buffer where the n	nessage should be stored.	
	length	Specifies the length in bytes c argument.	of the buffer pointed to by the <i>buffer</i>	
	flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:		
		MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next recv() or similar function will still return this data.	
		MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.	
		MSG_WAITALL	Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.	
	The recv() function returns the length of the message written to the buffer pointed to by the <i>buffer</i> argument. For message-based sockets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the <i>flags</i> argument, the excess bytes are discarded. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.			
	If the MCC MAITAIL flow is not got date will be returned only up to the and of the			

If the MSG\_WAITALL flag is not set, data will be returned only up to the end of the first message.

	If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, recv() blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, recv() fails and sets errno to EAGAIN or EWOULDBLOCK.		
USAGE	The recv() function is identiargument, and to read() if no	ical to recvfrom(3XNET) with a zero <i>address_len</i> of lags are used.	
	The select(3C) and poll(2) available to be received.	functions can be used to determine when data is	
RETURN VALUES	Upon successful completion, recv() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recv() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The recv() function will fail	if:	
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.	
	EBADF	The socket argument is not a valid file descriptor.	
	ECONNRESET	A connection was forcibly closed by a peer.	
	EFAULT	The <i>buffer</i> parameter can not be accessed or written.	
	EINTR	The recv() function was interrupted by a signal that was caught, before any data was available.	
	EINVAL	The MSG_OOB flag is set and no out-of-band data is available.	
	ENOTCONN	A receive is attempted on a connection-mode socket that is not connected.	
	ENOTSOCK	The socket argument does not refer to a socket.	
	EOPNOTSUPP	The specified flags are not supported for this socket type or protocol.	
	ETIMEDOUT	The connection timed out during connection establishment, or due to a transmission timeout on active connection.	
	The recv() function may fail if:		

recv(3XNET)

	EIO	An I/O erro the file syste	r occurred while reading from or writing to m.
	ENOBUFS	Insufficient r perform the	resources were available in the system to operation.
	ENOMEM	Insufficient i	nemory was available to fulfill the request.
	ENOSR		insufficient STREAMS resources available ation to complete.
ATTRIBUTES	See attributes(5) for descr	riptions of the	following attributes:
	ATTRIBUTE TYP	E	ATTRIBUTE VALUE
	MT-Level		MT-Safe
SEE ALSO			NET), select(3C), send(3XNET), down(3XNET), socket(3XNET),

NAME	recvfrom – receive a message from a socket			
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>			
		<pre>com(int socket, void *buffer ckaddr *address, socklen_t</pre>		
DESCRIPTION	The recvfrom() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.			
	The function takes	s the following arguments:		
	socket	Specifies the socket file descri	iptor.	
	buffer	Points to the buffer where the	e message should be stored.	
	length	Specifies the length in bytes c argument.	of the buffer pointed to by the <i>buffer</i>	
	flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:		
		MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next recvfrom() or similar function will still return this data.	
		MSG_OOB Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.		
		MSG_WAITALL	Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.	
	address	A null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.		
	address_len	Specifies the length of the so <i>address</i> argument.	ckaddr structure pointed to by the	
			the message written to the buffer ased sockets such as SOCK_DGRAM	

	and SOCK_SEQPACKET, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the <i>flags</i> argument, the excess bytes are discarded. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.		
	If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.		
	Not all protocols provide the source address for messages. If the <i>address</i> argument is not a null pointer and the protocol provides the source address of messages, the source address of the received message is stored in the sockaddr structure pointed to by the <i>address</i> argument, and the length of this address is stored in the object pointed to by the <i>address_len</i> argument.		
	If the actual length of the add structure, the stored address w	ress is greater than the length of the supplied sockaddr will be truncated.	
	If the <i>address</i> argument is not a null pointer and the protocol does not provide the source address of messages, the the value stored in the object pointed to by <i>address</i> is unspecified.		
	file descriptor, recvfrom() available at the socket and O_	t the socket and O_NONBLOCK is not set on the socket's blocks until a message arrives. If no messages are NONBLOCK is set on the socket's file descriptor, crno to EAGAIN or EWOULDBLOCK.	
USAGE	The select(3C) and poll(2) functions can be used to determine when data is available to be received.		
RETURN VALUES	Upon successful completion, recvfrom() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvfrom() returns 0. Otherwise the function returns -1 and sets errno to indicate the error.		
ERRORS	The recvfrom() function with	ill fail if:	
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.	
	EBADF	The <i>socket</i> argument is not a valid file descriptor.	
	ECONNRESET	A connection was forcibly closed by a peer.	
	EFAULT	The <i>buffer</i> , <i>address</i> or <i>address_len</i> parameter can not be accessed or written.	

			recvfrom(3XNET)	
	EINTR	A signal inte available.	errupted recvfrom() before any data was	
	EINVAL	The MSG_OOB flag is set and no out-of-band data is available.		
	ENOTCONN	A receive is that is not co	attempted on a connection-mode socket onnected.	
	ENOTSOCK	The socket ar	gument does not refer to a socket.	
	EOPNOTSUPP	The specified type.	d flags are not supported for this socket	
	ETIMEDOUT		ion timed out during connection nt, or due to a transmission timeout on ection.	
	The recvfrom() function m	ay fail if:		
	EIO	An I/O erro the file syste	r occurred while reading from or writing to em.	
	ENOBUFS	Insufficient 1 perform the	resources were available in the system to operation.	
	ENOMEM	Insufficient 1	memory was available to fulfill the request.	
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.		
ATTRIBUTES	See attributes(5) for descr	iptions of the	following attributes:	
	ATTRIBUTE TYP	Ξ	ATTRIBUTE VALUE	
	MT-Level		MT-Safe	
SEE ALSO	poll(2), recv(3XNET), recv sendmsg(3XNET), sendto(3 attributes(5)		select(3C) send(3XNET), down(3XNET), socket(3XNET),	

recvmsg(3XNET)				
NAME	recvmsg – receive a message from a socket			
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>			
	ssize_t recvms	<b>sg</b> (int <i>socket</i> , struct msgh	dr * <i>message</i> , int <i>flags</i> );	
DESCRIPTION	The recvmsg() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.			
	The function takes	s the following arguments:		
	socket	Specifies the socket file descri	iptor.	
	message	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The msg_flags member is ignored on input, but may contain meaningful values on output.		
	flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:		
		MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.	
		MSG_PEEK	Peeks at the incoming message.	
		MSG_WAITALL	Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.	
	The recymsg() function receives messages from unconnected or connected sockets and returns the length of the message.			
	sockets such as SC in a single operation MSG_PEEK is not as MSG_TRUNC is set stream-based sock	OCK_DGRAM and SOCK_SEQPAC on. If a message is too long to f set in the <i>flags</i> argument, the ex- in the msg_flags member of	cess bytes are discarded, and the msghdr structure. For ssage boundaries are ignored. In this	

		recvmsg(3AINET)		
	If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.			
	If no messages are available at the socket, and O_NONBLOCK is not set on the socket's file descriptor, recvmsg() blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, the recvmsg() function fails and sets errno to EAGAIN or EWOULDBLOCK.			
	In the msghdr structure, the msg_name and msg_namelen members specify the source address if the socket is unconnected. If the socket is connected, the msg_name and msg_namelen members are ignored. The msg_name member may be a null pointer if no names are desired or required. The <i>msg_iov</i> and <i>msg_iovlen</i> fields are used to specify where the received data will be stored. <i>msg_iov</i> points to an array of iovec structures; <i>msg_iovlen</i> must be set to the dimension of this array. In each iovec structure, the <i>iov_base</i> field specifies a storage area and the <i>iov_len</i> field gives its size in bytes. Each storage area indicated by <i>msg_iov</i> is filled with received data in turn until all of the received data is stored or all of the areas have been filled.			
	On successful completion, the msg_flags member of the message header is the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:			
	MSG_EOR End of record was received (if supported by the protocol).			
	MSG_00B Out-of-band data was received.			
	MSG_TRUNC	Normal data was truncated.		
	MSG_CTRUNC	Control data was truncated.		
USAGE	The select(3C) and poll(2) functions can be used to determine when data is available to be received.			
RETURN VALUES	Upon successful completion, recvmsg() returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvmsg() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.			
ERRORS	The recymsg() function will fail if:			
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.		
	EBADF	The <i>socket</i> argument is not a valid open file descriptor.		
	ECONNRESET A connection was forcibly closed by a peer.			

Networking Library Functions 421

recvmsg(3XNET)

	EFAULT	The <i>message</i> parameter, or storage pointed to by the <i>msg_name</i> , <i>msg_control</i> or <i>msg_iov</i> fields of the <i>message</i> parameter, or storage pointed to by the <i>iovec</i> structures pointed to by the <i>msg_iov</i> field can not be accessed or written.	
	EINTR	This function was interrupted by a signal before any data was available.	
	EINVAL	The sum of the iov_len values overflows an ssize_t. or the MSG_OOB flag is set and no out-of-band data is available.	
	EMSGSIZE	The msg_iovlen member of the msghdr structure pointed to by <i>message</i> is less than or equal to 0, or is greater than IOV_MAX.	
	ENOTCONN	A receive is attempted on a connection-mode socket that is not connected.	
	ENOTSOCK	The socket argument does not refer to a socket.	
	EOPNOTSUPP	The specified flags are not supported for this socket type.	
	ETIMEDOUT	The connection timed out during connection establishment, or due to a transmission timeout on active connection.	
	The recvmsg() function ma	y fail if:	
	EIO	An IO error occurred while reading from or writing t the file system.	
	ENOBUFS	Insufficient resources were available in the system to perform the operation.	
	ENOMEM	Insufficient memory was available to fulfill the request.	
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.	
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:	
	ATTRIBUTE TYP	E ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	poll(2), recv(3XNET), recv	<pre>/from(3XNET), select(3C), send(3XNET),</pre>	

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 cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ] Interfaces #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> #include <netdb.h> int res ninit(res state statp); void fp resstat(const res state state, 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 snendsigned**(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); Deprecated #include <sys/types.h> Interfaces #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> #include <netdb.h>

resolver(3RESOLV)				
	<pre>int res_initvoid););</pre>			
	<pre>int res_query(const char *dname, int class, int type, u_char *answer,</pre>			
	<pre>int res_search(const char *dname, int class, int type, u_char *answer,</pre>			
	<pre>int res_mkquery(int op, const char *dname, int class, int type, of char *data, int datalen, struct rrec *newrr, u_char *buf, int buflen);</pre>			
	<pre>int res_send(const u_char *msg, int msglen, u_char *answer, anslen);</pre>			
	void <b>herror</b> (const char	*s);		
DESCRIPTION	<b>These routines are used for making, sending, and interpreting query and reply messages with Internet domain name servers.</b>			
	<pre>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 compatability. 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-&gt;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, res_init() or res_ninit() has been called.</resolv.h></pre>			
	RES_DEBUG	Print debugging messages.		
	RES_AAONLY	Accept authoritative answers only. With this option, res_send() 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.		

		resolver(3RESOLV)
	RES_RECURSE	Set the recursion-desired bit in queries. This is the default. res_send() and res_nsend() do not do iterative queries and expect the name server to handle recursion.
	RES_DEFNAMES	If set, res_search() and res_nsearch() 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, res_search() and res_nsearch() search for host names in the current domain and in parent domains. See hostname(1). This option is used by the standard host lookup routine gethostbyname(3NSL). This option is enabled by default.
	RES_NOALIASES	This option turns off the user level aliasing feature controlled by the HOSTALIASES environment variable. Network daemons should set this option.
	RES_BLAST	If the RES_BLAST option is defined, resolver() queries will be sent to all servers. If the RES_BLAST option is not defined, but RES_ROTATE is, the list of nameservers are rotated according to a round-robin scheme. RES_BLAST overrides RES_ROTATE.
	RES_ROTATE	This option causes res_nsend() and res_send() to rotate the list of nameservers in statp->nsaddr_list or _res.nsaddr_list.
	RES_KEEPTSIG	This option causes res_nsendsigned() to leave the message unchanged after TSIG verification. Otherwise the TSIG record would be removed and the header would be updated.
res_ninit, res_init	The res_ninit() and res_init() 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 resolv.conf(4). If no server is configured, res_init() or res_ninit() will try to obtain name resolution services from the host on which it is running. The current domain name is defined by domainname(1M), or by the hostname if it is not specified in the configuration file. Use the environment variable LOCALDOMAIN 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 RES_OPTIONS environment variable to override certain internal resolver options. You can otherwise set them by changing fields in the statp /_res structure. Alternatively, they are inherited from the configuration file's options command. See	

resolver(3RESOLV)	
	resolv.conf(4) for information regarding the syntax of the RES_OPTIONS environment variable. Initialization normally occurs on the first call to one of the other resolver routines.
res_nquery, res_query	The res_nquery() and res_query() functions provides 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 message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. res_nquery() and res_query() return the length of the <i>answer</i> , or -1 upon error.
	The res_nquery() and res_query() 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.
res_nsearch, res_search	The res_nsearch() and res_search() routines make a query and await a response, just like like res_nquery() and res_query(). In addition, they implement the default and search rules controlled by the RES_DEFNAMES and RES_DNSRCH options. They return the length of the first successful reply which is stored in <i>answer</i> . On error, they reurn -1.
	The res_nsearch() and res_search() 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.
res_nmkquery, res_mkquery	These routines are used by res_nquery() and res_query(). The res_nmkquery() and res_mkquery() 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 QUERY, but can be any of the query types defined in <arpa nameser.h="">. The domain name for the query is given by <i>dname. newrr</i> is currently unused but is intended for making update messages.</arpa>
res_nsend, res_send, res_nsendsigned	The res_nsend(), res_send(), and res_nsendsigned() routines send a preformatted query that returns an <i>answer</i> . The routine calls res_ninit() or res_init(). If RES_INIT is not set, the routine sends the query to the local name server and handles timeouts and retries. Additionally, the res_nsendsigned() 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.
	The res_nsend() and res_send() 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.			
res_npquery	The function $res_npquery()$ prints out the query and any answer in <i>msg</i> on <i>fp</i> .			
fp_resstat	The function fp_resstat() prints out the active flag bits in statp->options preceded by the text ";; res options:" on <i>file</i> .			
res_hostalias	The function res_hostalias() 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. res_hostalias() stores the result in <i>buf</i> .			
res_nclose	The res_nclose() function closes any open files referenced through <i>statp</i> .			
dn_comp	dn_comp() compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i> . dn_comp() 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> .			
	<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> .			
	A side effect of calling dn_comp() is to update the list of pointers for labels inserted into the message by dn_comp() as the name is compressed. If <i>dnptrs</i> is NULL, names are not compressed. If <i>lastdnptr</i> is NULL, dn_comp() does not update the list of labels.			
dn_expand	$dn\_expand()$ 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> dn_expand() returns the size of the compressed name, or $-1$ if there was an error.			
hstrerror, herror	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 <netdb.h>: #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 */ The herror() function writes a message to the diagnostic output consisting of the string parameters, the constant string ": ", and a message corresponding to the value</netdb.h>			
	of <i>h_errno</i> .			

### resolver(3RESOLV)

The hstrerror() function returns a string, which is the message text that corresponds to the value of the *err* parameter.

FILES /etc/resolv.conf

#### ATTRIBUTES

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

	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWcsl (32-bit)	
		SUNWcslx (64-bit)	
	Interface Stability	Standard BIND 8.2.4	
	MT-Level	Unsafe for Deprecated Interfaces; MT-Safe for all others.	
EE ALSO	<pre>domainname(1M), in.named(1M), gethostbyname(3NSL), libresolv(3LIB), resolv.conf(4), attributes(5)</pre>		
	<ul> <li>Lottor, M. <i>RFC 1033, Domain Administrators Operations Guide</i>. Network Working Group. November 1987.</li> <li>Mockapetris, Paul. <i>RFC 1034, Domain Names - Concepts and Facilities</i>. Network Work Group. November 1987.</li> <li>Mockapetris, Paul. <i>RFC 1035, Domain Names - Implementation and Specification</i>. Netw Working Group. November 1987.</li> <li>Partridge, Craig. <i>RFC 974, Mail Routing and the Domain System</i>. Network Working Group. January 1986.</li> <li>Stahl, M. <i>RFC 1032, Domain Administrators Guide</i>. Network Working Group. Novem 1987.</li> <li>Vixie, Paul, Dunlap, Kevin J., Karels, Michael J. <i>Name Server Operations Guide for BIN</i> Internet Software Consortium, 1996.</li> </ul>		
NOTES	<b>NOTES</b> When the caller supplies a work buffer, for example the <i>answer</i> buffer arguments nsend() or res_send(), the buffer should be aligned on an eight b boundary. Otherwise, an error such as a SIGBUS may result.		
		,	

428 man pages section 3: Networking Library Functions • Last Revised 7 Sep 2001

NAME	rexec, rexec_af – return stream to a remote command			
SYNOPSIS	cc [ flag ] filelsocket -lnsl [ library ]			
	<pre>int rexec(char **ahost, unsigned short inport, const char *user, const char *passwd, const char *cmd, int *fd2p);</pre>			
	<pre>int rexec_af(char **ahost, unsigned short inport, const char *user, const char *passwd, const char *cmd, int *fd2p, int af);</pre>			
DESCRIPTION	<pre>rexec() and rexec_af() look up the hostahost using getipnodebyname(3SOCKET), returning -1 if the host does not exist. Otherwise ahost is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the user's .netrc file in his home directory is searched for appropriate information. If all this fails, the user is prompted for the information.</pre>			
	The difference between rexec() and rexec_af() is that while rexec() always returns a socket of the AF_INET address family, with rexec_af() the application can choose which type of address family the socket returned should be. rexec_af() supports both AF_INET and AF_INET6 address families.			
	The port <i>inport</i> specifies which well-known DARPA Internet port to use for the connection. The protocol for connection is described in detail in in.rexecd(1M).			
	If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as its standard input and standard output. If $fd2p$ is non-zero, then an auxiliary channel to a control process will be setup, and a file descriptor for it will be placed in * $fd2p$ . The control process will return diagnostic output (file descriptor 2, the standard error) from the command on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If $fd2p$ is 0, then the standard error (file descriptor 2 of the remote command) will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.			
RETURN VALUES	If rexec() succeeds, a file descriptor number, which is a socket of type SOCK_STREAM and address family AF_INET is returned by the routine. * <i>ahost</i> is set to the standard name of the host, and if <i>fd2p</i> is not NULL, a file descriptor number is placed in * <i>fd2p</i> which represents the command's standard error stream.			
	If rexec_af() succeeds, the routine returns a filed descriptor number, which is a socket of type SOCK_STREAM and of address family type AF_INET or AF_INET, as determined by the value of the <i>af</i> parameter that the caller passes in.			
	If either rexec() or rexec_af() fails, -1 is returned.			
ATTRIBUTES	See attributes (5) for descriptions of the following attributes:			

# rexec(3SOCKET)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE		
	MT-Level	Unsafe		
SEE ALSO	<pre>in.rexecd(1M), gethostbyname(3NSL), getipnodebyname(3SOCKET), getservbyname(3SOCKET), socket(3SOCKET), attributes(5)</pre>			
NOTES	There is no way to specify options to the socket () call that rexec() or rexec_af() makes.			
	This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.			

NAME	rpc – library routir	nes for remote procedure calls		
SYNOPSIS	<pre>cc [ flag ] filelnsl [ library ] #include <rpc rpc.h=""> #include <netconfig.h></netconfig.h></rpc></pre>			
DESCRIPTION	These routines allow C language programs to make procedure calls on other machines across a network. First, the client sends a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.			
	structure also requ	outines require the header <rpc rpc.h="">. Routines that take a netconfig also require that <netconfig.h> be included. Applications using RPC and ines should be linked with the libnsl library.</netconfig.h></rpc>		
Multithread Considerations				
	When used in multithreaded applications, server-side routines are usually Unsafe. In this implementation the service transport handle, SVCXPRT contains a single data area for decoding arguments and encoding results. See rpc_svc_create(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 rpc_svc_calls(3NSL).			
Nettyp	Some of the high-level RPC interface routines take a <i>nettype</i> string as one of the parameters (for example, clnt_create(), svc_create(), rpc_reg(), rpc_call()). This string defines a class of transports which can be used for a particular application.			
	<i>nettype</i> can be one of the following:			
token names in the NETPATH environment		Choose from the transports which have been indicated by their token names in the NETPATH environment variable. If NETPATH is unset or NULL, it defaults to visible. netpath is the default <i>nettype</i> .		
	visible Choose the transports which have the visible flag (v) set /etc/netconfig file.			
	circuit_v This is same as visible except that it chooses only the connection oriented transports (semantics tpi_cots or tpi_cots_ord) from the entries in the /etc/netconfig file.			

rpc(3NSL)

	datagram_v	This is same as visible except that it chooses only the connectionless datagram transports (semantics tpi_clts) from the entries in the /etc/netconfig file.		
	circuit_n	This is same as netpath except that it chooses only the connection oriented datagram transports (semantics tpi_cots or tpi_cots_ord). This is same as netpath except that it chooses only the connectionless datagram transports (semantics tpi_clts).		
	datagram_n			
	udp	This refers to Internet UDP.		
	tcp	This refers to Internet TCP.		
		it defaults to netpath. The transports are tried in left to right order priable or in top to down order in the /etc/netconfig file.		
Derived Types	In a 64-bit environ	environment, the derived types are defined as follows:		
	typedef	uint32_t	<pre>rpcprog_t;</pre>	
	typedef	uint32_t	<pre>rpcvers_t;</pre>	
	typedef	uint32_t	<pre>rpcproc_t;</pre>	
	typedef	uint32_t	<pre>rpcprot_t;</pre>	
	typedef	uint32_t	<pre>rpcport_t;</pre>	
	typedef	int32_t	<pre>rpc_inline_t;</pre>	
	In a 32-bit environ	ment, the derived types are defined as follows:		
	typedef	unsigned long	<pre>rpcprog_t;</pre>	
	typedef	unsigned long	<pre>rpcvers_t;</pre>	
	typedef	unsigned long	<pre>rpcproc_t;</pre>	
	typedef	unsigned long	<pre>rpcprot_t;</pre>	
	typedef	unsigned long	<pre>rpcport_t;</pre>	
	typedef	long	<pre>rpc_inline_t;</pre>	
Data Structures	Some of the data s	structures used by the RPC package a	re shown below	
The AUTH	union des block	_		
Structure	struct { u_int32 u_int32	high;		

432 man pages section 3: Networking Library Functions • Last Revised 5 Jun 2001

```
} key;
               char c[8];
               };
               typedef union des_block des_block;
               extern bool t xdr des block();
               /*
               * Authentication info. Opaque to client.
               */
               struct opaque_auth {
                                           /* flavor of auth */
                      enum t oa flavor;
                                            /* address of more auth stuff */
                      caddr t oa base;
                                            /* not to exceed MAX AUTH BYTES */
                      uint t oa length;
               };
               /*
               * 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 marshal)();
                                                     /* nextverf & serialize */
                              int(*ah validate)();
                                                    /* validate verifier */
                              int(*ah refresh)();
                                                    /* refresh credentials */
                                                   /* destroy this structure */
                              void(*ah_destroy)();
                      } *ah_ops;
                      caddr_t ah_private;
               } AUTH;
 The CLIENT
              /*
 Structure
               * 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 */
                                                             /* abort a call */
                            void (*cl_abort)();
                                                          /* get specific error code */
                            void (*cl geterr)();
                            bool t (*cl freeres)();
                                                            /* frees results */
                            void (*cl_destroy)();
                                                           /* destroy this structure */
                            bool_t (*cl_control)();
                                                            /* the ioctl( ) of rpc */
                                                            /* set rpc level timers */
                            int (*cl settimers)();
                           } *cl ops;
                                                                    /* private stuff */
                           caddr_t
                                     cl_private;
                                                             /* network identifier */
                           char
                                     *cl netid;
                                     *cl_tp;
                                                                   /* device name */
                           char
               } CLIENT;
The SVCXPRT
              enum xprt_stat {
 Structure
              XPRT_DIED,
              XPRT MOREREQS,
               XPRT_IDLE
              };
               /*
               *
                  Server side transport handle
```

	*/	
	typedef struct {	
	int xp_fd;	<pre>/* file descriptor for the</pre>
	ushort_t xp_port;	/* obsolete */
	struct xp_ops {	
		receive incoming requests */
		( ); /* get transport status */
	<pre>bool_t (*xp_getargs)();</pre>	/* get arguments */
	<pre>bool_t (*xp_reply)();</pre>	/* send reply */
	bool_t (*xp_freeargs)();	/* free mem allocated
	unid (turn doct rout) ().	for args */ /* destroy this struct */
	<pre></pre>	/ descroy this struct "/
	int xp_addrlen;	<pre>/* length of remote addr.</pre>
	ine np_uudrien,	Obsolete */
	char *xp tp;	/* transport provider device
		name */
	char *xp netid;	/* network identifier */
	struct netbuf xp_ltaddr;	/* local transport address */
	struct netbuf xp_rtaddr;	<pre>/* remote transport address */</pre>
	<pre>char xp_raddr[16];</pre>	/* remote address. Obsolete */
	<pre>struct opaque_auth xp_verf;</pre>	/* raw response verifier */
	<pre>caddr_t xp_p1;</pre>	<pre>/* private: for use</pre>
		by svc ops */
	caddr_t xp_p2;	/* private: for use
	andda t ym n2.	by svc ops */
	<pre>caddr_t xp_p3;</pre>	/* private: for use by svc lib */
	int xp type	/* transport type */
	<pre>svcxprt;</pre>	/ cranspore cype /
	, ,	
The svc_reg	struct svc_req {	
Structure	rpcprog_t rq_prog; /* se	ervice program number */
	rpcvers_t rq_vers; /* se	ervice protocol version */
	<pre>rpcproc_t rq_proc;</pre>	e desired procedure */
	caddr_t rq_clntcred; /* re	
		sociated transport */
		1 /
	};	
The XDR	/*	
Structure	* XDR operations.	
beruceure	* XDR ENCODE causes the type to be	encoded into the stream
	* XDR DECODE causes the type to be	
		he space allocated by an XDR DECODE
	* request.	
	*/	
	enum xdr_op {	
	XDR_ENCODE=0,	
	XDR_DECODE=1,	
	XDR_FREE=2	
	<pre>};</pre>	
	/*	mit of external data
	* This is the number of bytes per u */	mit of external data.
	#define BYTES PER XDR UNIT (4)	
	#define RNDUP(x) ((((x) + BYTES_PEF	2 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
                    \star 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 {
                                        /* operation; fast additional param */
                   enum xdr op x op;
                   struct xdr_ops {
                                                    /* get long from underlying stream */
                   bool t
                               (*x getlong)();
                                                     /* put long to underlying stream */
                   bool t
                               (*x putlong)();
                   bool t
                              (*x getbytes)();
                                                     /* get bytes from underlying stream */
                   bool t
                              (*x putbytes)(); /* put bytes to underlying stream */
                              (*x_getpostn)(); /* returns bytes off from beginning */
                   uint_t
                                                    /* reposition the stream */
                               (*x_setpostn)();
                   bool t
                   rpc_inline_t *(*x_inline)();
void (*x_destroy)();
                                                     /* buf quick ptr to buffered data */
                                                    /* free privates of this xdr stream */
                                                    /* changed/retrieve client object info*/
                              (*x_control)();
                   bool t
                   bool_t (*x_getint32)(); /* get int from underlying stream */
bool_t (*x_putint32)(); /* get int from underlying stream */
                               (*x putint32)();
                                                    /* put int to underlying stream */
                   bool t
                   } *x ops;
                   caddr t
                               x public;
                                                     /* users' data */
                               x_priv
x_base;
                                                     /* pointer to private data */
                   caddr t
                                                     /* private used for position info */
                   caddr_t
                   int
                                x handy;
                                                     /* extra private word */
                   XDR:
Index to Routines
                   The following table lists RPC routines and the manual reference pages on which they
                   are described:
                   RPC Routine
                                                          Manual Reference Page
                                                          rpc clnt auth(3NSL)
                   auth destroy
                   authdes create
                                                          rpc soc(3NSL)
                   authdes getucred
                                                          secure rpc(3NSL)
                   authdes seccreate
                                                          secure rpc(3NSL)
                   authnone create
                                                          rpc clnt auth(3NSL)
                                                          rpc clnt auth(3NSL)
                   authsys create
```

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) rpc clnt calls(3NSL) clnt geterr 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) rpc clnt create(3NSL) clnt spcreateerror rpc clnt calls(3NSL) clnt sperrno clnt sperror rpc clnt calls(3NSL) clnt tli create rpc clnt create(3NSL) rpc clnt create(3NSL) clnt\_tp\_create clnt udpcreate rpc soc(3NSL) rpc\_clnt\_create(3NSL) clnt vc create clntraw create rpc soc(3NSL) clnttcp create rpc soc(3NSL) clntudp\_bufcreate rpc\_soc(3NSL) get myaddress rpc soc(3NSL) secure rpc(3NSL) getnetname host2netname secure rpc(3NSL) key decryptsession secure rpc(3NSL) key\_encryptsession secure rpc(3NSL)

key_gendes	secure_rpc(3NSL)
key_setsecret	secure_rpc(3NSL)
netname2host	secure_rpc(3NSL)
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	<pre>rpc_rac(3RAC)</pre>
rac_poll	<pre>rpc_rac(3RAC)</pre>
rac_recv	<pre>rpc_rac(3RAC)</pre>
rac_send	<pre>rpc_rac(3RAC)</pre>
registerrpc	rpc_soc(3NSL)
rpc_broadcast	<pre>rpc_clnt_calls(3NSL)</pre>
rpc_broadcast_exp	<pre>rpc_clnt_calls(3NSL)</pre>
rpc_call	<pre>rpc_clnt_calls(3NSL)</pre>
rpc_reg	<pre>rpc_svc_calls(3NSL)</pre>
svc_create	<pre>rpc_svc_create(3NSL)</pre>
svc_destroy	<pre>rpc_svc_create(3NSL)</pre>
<pre>svc_dg_create</pre>	<pre>rpc_svc_create(3NSL)</pre>
<pre>svc_dg_enablecache</pre>	<pre>rpc_svc_calls(3NSL)</pre>
<pre>svc_fd_create</pre>	<pre>rpc_svc_create(3NSL)</pre>
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	<pre>rpc_svc_calls(3NSL)</pre>
<pre>svc_getrpccaller</pre>	<pre>rpc_svc_calls(3NSL)</pre>
svc_raw_create	<pre>rpc_svc_create(3NSL)</pre>
I	

Networking Library Functions 437

svc_reg	<pre>rpc_svc_calls(3NSL)</pre>
svc_register	rpc_soc(3NSL)
svc_run	rpc_svc_reg(3NSL)
svc_sendreply	rpc_svc_reg(3NSL)
<pre>svc_tli_create</pre>	<pre>rpc_svc_create(3NSL)</pre>
<pre>svc_tp_create</pre>	<pre>rpc_svc_create(3NSL)</pre>
svc_unreg	<pre>rpc_svc_calls(3NSL)</pre>
svc_unregister	rpc_soc(3NSL)
svc_vc_create	<pre>rpc_svc_create(3NSL)</pre>
svcerr_auth	<pre>rpc_svc_err(3NSL)</pre>
svcerr_decode	<pre>rpc_svc_err(3NSL)</pre>
svcerr_noproc	<pre>rpc_svc_err(3NSL)</pre>
svcerr_noprog	<pre>rpc_svc_err(3NSL)</pre>
svcerr_progvers	rpc_svc_err(3NSL)
svcerr_systemerr	<pre>rpc_svc_err(3NSL)</pre>
svcerr_weakauth	<pre>rpc_svc_err(3NSL)</pre>
svcfd_create	rpc_soc(3NSL)
svcraw_create	rpc_soc(3NSL)
svctcp_create	rpc_soc(3NSL)
svcudp_bufcreate	rpc_soc(3NSL)
svcudp_create	rpc_soc(3NSL)
user2netname	secure_rpc(3NSL)
xdr_accepted_reply	rpc_xdr(3NSL)
xdr_authsys_parms	rpc_xdr(3NSL)
xdr_authunix_parms	rpc_soc(3NSL)
xdr_callhdr	rpc_xdr(3NSL)
xdr_callmsg	rpc_xdr(3NSL)
xdr_opaque_auth	rpc_xdr(3NSL)
xdr_rejected_reply	rpc_xdr(3NSL)
xdr_replymsg	rpc_xdr(3NSL)
xprt_register	<pre>rpc_svc_calls(3NSL)</pre>

438 man pages section 3: Networking Library Functions • Last Revised 5 Jun 2001

xprt\_unregister rpc svc calls(3NSL) FILES /etc/netconfig ATTRIBUTES See attributes(5) for descriptions of the following attributes: ATTRIBUTE TYPE ATTRIBUTE VALUE MT-Level MT-Safe with exceptions **SEE ALSO** getnetconfig(3NSL), getnetpath(3NSL), rpc\_clnt\_auth(3NSL), rpc clnt calls(3NSL), rpc clnt create(3NSL), rpc svc calls(3NSL), rpc svc create(3NSL), rpc svc err(3NSL), rpc svc reg(3NSL), rpc\_xdr(3NSL), rpcbind(3NSL), secure\_rpc(3NSL), threads(3THR), xdr(3NSL), netconfig(4), rpc(4), attributes(5), environ(5)

## rpcbind(3NSL)

NAME	rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>struct rpcblist *rpcb_getmaps(const struct netconfig *nnetconf,</pre>
	<pre>bool_t rpcb_getaddr(const rpcprog_t prognum, const rpcvers_t     versnum, const struct netconfig *netconf, struct netbuf *ssvcaddr,     const char *host);</pre>
	<pre>bool_t rpcb_gettime(const char *host, time_t *timep);</pre>
	<pre>enum clnt_stat rpcb_rmtcall(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);</pre>
	<pre>bool_t rpcb_set(const rpcprog_t prognum, const rpcvers_t versnum,</pre>
	<pre>bool_t rpcb_unset(const rpcprog_t prognum, const rpcvers_t versnum,</pre>
DESCRIPTION	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).
Routines	<pre>rpcb_getmaps() An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on host. It uses the transport specified through netconf to contact the remote rpcbind service on host. This routine will return NULL, if the remote rpcbind could not be contacted.</pre>
	<pre>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 or that the RPC system failed to contact the remote rpcbind service. In the latter case, the global variable rpc_createerr contains the RPC status. See rpc_clnt_create(3NSL).</pre>
	<pre>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.</pre>

		-F()
	RPC call on your behalf to a procedure of should correspond to a connectionless to modified to the server's address if the pr	_
	This procedure should normally be used allows programs to do lookup and call,	l for a "ping" and nothing else. This routine all in one step.
	Note: Even if the server is not running <b>r</b> messages to the caller. In such a case, the	
	Note: rpcb_rmtcall() is only availab	le for connectionless transports.
	service. The value of <i>nc_netid</i> must correct defined by the netconfig database. This	and <i>svcaddr</i> on the machine's rpcbind spond to a network identifier that is routine returns TRUE if it succeeds, FALSE _svc_calls (3NSL). If there already exists
	the triple [prognum, versnum, all-transpor	<i>tid]</i> and the address on the machine's <i>b_unset()</i> destroys all mapping between <i>ts</i> ] and the addresses on the machine's RUE if it succeeds, FALSE otherwise. Only er can destroy the mapping. See also
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
SEE ALSO	rpcbind(1M), rpcinfo(1M), rpc_clnt_ rpc_svc_calls(3NSL), attributes(5)	calls(3NSL),rpc_clnt_create(3NSL),

## rpc\_clnt\_auth(3NSL)

NAME	rpc_clnt_auth, auth_destroy, authnone_create, authsys_create, authsys_create_default – library routines for client side remote procedure call authentication
SYNOPSIS	<pre>void auth_destroy(AUTH *auth);</pre>
	AUTH <b>*authnone_create(</b> void););
	AUTH <b>*authsys_create</b> (const cha r* <i>host</i> , const uid_t <i>uid</i> , const gid_t gid, const int len, const gid_t *aup_gids);
	AUTH <b>*authsys_create_default</b> void););
DESCRIPTION	These routines are part of the RPC library that allows C language programs to make procedure calls on other machines across the network, with desired authentication.
	These routines are normally called after creating the CLIENT handle. The cl_auth field of the CLIENT structure should be initialized by the AUTH structure returned by some of the following routines. The client's authentication information is passed to the server when the RPC call is made.
	Only the NULL and the SYS style of authentication is discussed here. For the DES style authentication, please refer to secure_rpc(3NSL).
	The NULL and SYS style of authentication are safe in multithreaded applications. For the MT-level of the DES style, see its pages.
Routines	The following routines require that the header <rpc rpc.h=""> be included (see rpc(3NSL) for the definition of the AUTH data structure).</rpc>
	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>auth_destroy() A function macro that destroys the authentication information associated with auth. Destruction usually involves deallocation of private data structures. The use of auth is undefined after calling auth_destroy().</pre>
	authnone_create() Create and return an RPC authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.
	<pre>authsys_create() Create and return an RPC authentication handle that contains AUTH_SYS authentication information. The parameter <i>host</i> is the name of the machine on which the information was created; <i>uid</i> is the user's user ID; <i>gid</i> is the user's current group ID; <i>len</i> and <i>aup_gids</i> refer to a counted array of groups to which the user belongs.</pre>
	authsys_create_default Call authsys_create() with the appropriate parameters.

## rpc\_clnt\_auth(3NSL)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO rpc(3NSL), rpc\_clnt\_calls(3NSL), rpc\_clnt\_create(3NSL), secure\_rpc(3NSL), attributes(5)

## rpc\_clnt\_calls(3NSL)

NAME	rpc_clnt_calls, clnt_call, clnt_send, 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
SYNOPSIS	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>enum clnt_stat clnt_call(CLIENT *clnt, const rpcproc_t procnum,</pre>
	<pre>enum clnt_stat clnt_send (CLIENT *clnt, const u_long procnum, const xdrproc_t proc, const caddr_t in);</pre>
	<pre>bool_t clnt_freeres(CLIENT *clnt, const xdrproc_t outproc, caddr_t</pre>
	<pre>void clnt_geterr(const CLIENT *clnt, struct rpc_err *errp);</pre>
	<pre>void clnt_perrno(const enum clnt_stat stat);</pre>
	<pre>void clnt_perror(const CLIENT *clnt, const char *s);</pre>
	<pre>char *clnt_sperrno(const enum clnt_stat stat);</pre>
	<pre>char *clnt_sperror(const CLIENT *clnt, const char *s);</pre>
	<pre>enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_tinproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char *nettype);</pre>
	<pre>enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_txargs, caddr_t argsp, const xdrproc_txresults, caddr_t resultsp, const resultproc_t eachresult, const int inittime, const int waittime, const char *nettype);</pre>
	<pre>enum clnt_stat rpc_call(const char *host, const rpcprog_t prognum,</pre>
DESCRIPTION	RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service and then sends back a reply.
	The clnt_call(), rpc_call(), and rpc_broadcast() routines handle the client side of the procedure call. The remaining routines deal with error handling.
	Some of the routines take a CLIENT handle as one of the parameters. A CLIENT handle can be created by an RPC creation routine such as clnt_create(). See rpc_clnt_create(3NSL).

444 man pages section 3: Networking Library Functions • Last Revised 1 Aug 2001

These routines are safe for use in multithreaded applications. CLIENT handles can be shared between threads; however, in this implementation requests by different threads are serialized. In other words, the first request will receive its results before the second request is sent.

Routines S

See rpc(3NSL) for the definition of the CLIENT data structure.

## clnt\_call()

A function macro that calls the remote procedure *procnum* associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as clnt\_create(). See rpc\_clnt\_create(3NSL). The parameter *inproc* is the XDR function used to encode the procedure's parameters, and *outproc* is the XDR function 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). *tout* is the time allowed for results to be returned, which is overridden by a time-out set explicitly through clnt\_control(). See rpc\_clnt\_create(3NSL).

If the remote call succeeds, the status returned is RPC\_SUCCESS. Otherwise, an appropriate status is returned.

#### clnt send()

Use the clnt send() function to call a remote asynchronous function.

The clnt\_send() function calls the remote function procnum() associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as clnt\_create(). See rpc\_clnt\_create(3NSL). The parameter *proc* is the XDR function used to encode the procedure's parameters. The parameter *in* is the address of the procedure's argument(s).

By default, the blocking I/O mode is used. See the clnt\_control(3NSL) man page for more information on I/O modes.

The clnt\_send() function does not check if the program version number supplied to clnt\_create() is registered with the rpcbind service. Use clnt\_create\_vers() instead of clnt\_create() to check on incorrect version number registration.clnt\_create\_vers() will return a valid handle to the client only if a version within the range supplied to clnt\_create\_vers() is supported by the server.

RPC\_SUCCESS is returned when a request is successfully delivered to the transport layer. This does not mean that the request was received. If an error is returned, use the clnt\_getterr() routine to find the failure status or the clnt\_perrno() routine to translate the failure status into error messages.

## clnt\_freeres()

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

#### clnt\_geterr()

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

### clnt\_perrno()

Prints a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. It is normally used after a procedure call fails for a routine for which a client handle is not needed, for instance rpc\_call()

#### clnt perror()

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

#### clnt sperrno()

Takes the same arguments as clnt\_perrno(), but instead of sending a message to the standard error indicating why an RPC call failed, returns a pointer to a string that contains the message.

clnt\_sperrno() is normally used instead of clnt\_perrno() when the program does not have a standard error, as a program running as a server quite likely does not.clnt\_sperrno() is also used if the programmer does not want the message to be output with printf(), or if a message format different than that supported by clnt\_perrno() is to be used. See printf(3C). Unlike clnt\_sperror() and clnt\_spcreaterror(), clnt\_sperrno() does not return a pointer to static data. Therefore, the result is not overwritten on each call. See rpc\_clnt\_create(3NSL).

### clnt\_sperror()

Similar to clnt\_perror(), except that like clnt\_sperror(), it returns a string instead of printing to standard error. However, clnt\_sperror() does not append a newline at the end of the message.

clnt\_sperror() returns a pointer to a buffer that is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.

## rpc\_broadcast()

Similar to rpc\_call(), except that 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. The 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).

### rpc broadcast exp()

Similar to rpc\_broadcast(), except that the initial timeout, *inittime* and the maximum timeout, *waittime*, are specified in milliseconds.

*inittime* is the initial time that rpc\_broadcast\_exp() waits before resending the request. After the first resend, the retransmission interval increases exponentially until it exceeds *waittime*.

### rpc call()

Calls 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 it returns an appropriate status. Use the clnt perrno() routine to translate failure status into error messages.

The rpc\_call() function 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
Architecture	All
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Evolving
MT-Level	MT-Safe

#### 

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_spcreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr, clnt_door_create – library routines for dealing with creation and manipulation of CLIENT handles
SYNOPSIS	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>bool_t clnt_control(CLIENT *clnt, const uint_t req, char *info);</pre>
	CLIENT <b>*clnt_create</b> (const char <b>*</b> <i>host</i> , const rpcprog_t <i>prognum</i> , const rpcvers_t <i>versnum</i> , const char <b>*</b> <i>nettype</i> );
	<pre>CLIENT *clnt_create_timed(const char *host, const rpcprog_t     prognum, const rpcvers_t versnum, const nettype, const struct     timeval *timetout);</pre>
	<pre>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);</pre>
	<pre>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);</pre>
	<pre>void clnt_destroy(CLIENT *clnt);</pre>
	<pre>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 recsz);</pre>
	<pre>void clnt_pcreateerror(const char *s);</pre>
	<pre>CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t</pre>
	char <b>*clnt_spcreateerror</b> (const char <b>*</b> <i>s</i> );
	<pre>CLIENT *clnt_tli_create(const int fildes, const struct netconfig     *netconf, const struct netbuf *svcaddr, const rpcprog_t prognum,     const rpcvers_t versnum, const uint_t sendsz, const uint_t     recsz);</pre>
	CLIENT <b>*clnt_tp_create</b> (const char <b>*</b> <i>host</i> , const rpcprog_t <i>prognum</i> , const rpcvers_t <i>versnum</i> , const struct netconfig <b>*</b> <i>netconf</i> );
	<pre>CLIENT *clnt_tp_create_timed(const char *host, const rpcprog_t     prognum, const rpcvers_t versnum, const struct netconfig     *netconf, const struct timeval *timeout);</pre>
	<pre>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 recsz);</pre>
	struct rpc_createerr rpc_createerr

	<pre>CLIENT *clnt_door_create(const rpcprog_t prognum, const rpcvers_t</pre>
DESCRIPTION	RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.
	These routines are MT-Safe. In the case of multithreaded applications, the -mt option must be specified on the command line at compilation time. When the -mt option is specified, rpc_createerr() becomes a macro that enables each thread to have its own rpc_createerr(). See threads(3THR).
Routines	See rpc(3NSL) for the definition of the CLIENT data structure.
	<pre>clnt_control() A function macro to change or retrieve various information about a client object. req indicates the type of operation, and info is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of req and their argument types and what they do are:</pre>
	CLSET_TIMEOUT struct timeval * set total timeout
	CLGET_TIMEOUT struct timeval * get total timeout
	If the timeout is set using clnt_control(), the timeout argument passed by clnt_call() is ignored in all subsequent calls. If the timeout value is set to 0, clnt_control() immediately returns RPC_TIMEDOUT. Set the timeout parameter to 0 for batching calls.
	<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</pre>
	<pre>chose_value reported by the program by version     number associated with the     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</pre>

	The following operations are valid for connection-oriented transports of
	CLSET_IO_MODE rpciomode_t* set the IO mode used
	to send one-way requests. The argument for this operation
	can be either:
	- RPC_CL_BLOCKING all sending operations block
	until the underlying transport protocol has
	accepted requests. If you specify this argument
	you cannot use flush and getting and setting buffer
	size is meaningless.
	- RPC_CL_NONBLOCKING sending operations do not
	block and return as soon as requests enter the buffer. You can now use non-blocking I/O. The requests in the
	buffer are pending. The requests are sent to
	the server as soon as a two-way request is sent
	or a flush is done. You are responsible for flushing
	the buffer. When you choose RPC CL NONBLOCKING argument
	you have a choice of flush modes as specified by
	CLSET FLUSH MODE.
(	CLGET_IO_MODE rpciomode_t* get the current IO mode
	CLSET_FLUSH_MODE rpcflushmode_t* set the flush mode.
	The flush mode can only be used in non-blocking I/O mode.
	The argument can be either of the following:
	- RPC CL BESTEFFORT FLUSH: All flushes send requests
	in the buffer until the transport end-point blocks.
	If the transport connection is congested, the call
	returns directly.
	- RPC_CL_BLOCKING_FLUSH: Flush blocks until the
	underlying transport protocol accepts all pending
	requests into the queue.
	CLGET_FLUSH_MODE rpcflushmode_t* get the current flush mode.
	CLFLUSH rpcflushmode_t flush the pending requests.
	This command can only be used in non-blocking I/O mode.
	The flush policy depends on which of the following
	parameters is specified:
	- RPC_CL_DEFAULT_FLUSH, or NULL: The flush is done
	according to the current flush mode policy
	(see CLSET_FLUSH_MODE option). - RPC_CL_BESTEFFORT_FLUSH: The flush tries
	to send pending requests without blocking; the call
	returns directly. If the transport connection is
	congested, this call could return without the request
	being sent.
	- RPC_CL_BLOCKING_FLUSH: The flush sends all pending
	requests. This call will block until all the requests
	have been accepted by the transport layer.
(	CLSET CONNMAXREC SIZE int* set the buffer size.
	It is not possible to dynamically
	resize the buffer if it contains data.
	The default size of the buffer is 16 kilobytes.
	CLGET_CONNMAXREC_SIZE int* get the current size of the
	buffer
	CLGET_CURRENT_REC_SIZE int* get the size of
	the pending requests stored in the buffer. Use of this
	command is only recommended when you are in non-blocking
	I/O mode. The current size of the buffer is always zero

450 man pages section 3: Networking Library Functions • Last Revised 1 Aug 2001

used in this mode.

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.

#### clnt\_create()

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 that 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)).

## clnt\_create\_timed()

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.

#### clnt create vers()

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 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.

## clnt\_create\_vers\_timed()

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.

### clnt\_destroy()

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)).

### clnt\_dg\_create()

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 *recevsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

#### clnt\_pcreateerror()

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.

#### clnt raw create()

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().

### clnt\_spcreateerror()

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.

#### clnt tli create()

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. *fildes* 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 *fildes* is RPC\_ANYFD and *netconf* is NULL, a RPC\_UNKNOWNPROTO error is set. If *fildes* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), clnt\_tli\_create() calls appropriate client creation routines. This routine returns NULL if it fails. The clnt\_pcreateerror() routine can be used to print the reason for failure. The remote rpcbind service (see rpcbind(1M)) is not consulted for the address of the remote service.

#### clnt tp create()

Like clnt\_create() except clnt\_tp\_create() tries only one transport
specified through netconf.

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

#### clnt tp create timed()

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.

### clnt\_vc\_create()

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.

```
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.

clnt\_door\_create()

This routine creates an RPC client handle over doors for the given program *prognum* and version *versnum*. Doors is a transport mechanism that facilitates fast data transfer between processes on the same machine. The user may set the size of the send buffer with the parameter *sendsz*. If *sendsz* is 0, the corresponding default buffer size is 16 Kbyte. The clnt\_door\_create() routine returns NULL if it fails and sets a value for rpc\_createerr.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	All
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO rpcbind(1M), rpc(3NSL), rpc\_clnt\_auth(3NSL), rpc\_clnt\_calls(3NSL), rpc\_svc\_create(3NSL), svc\_raw\_create(3NSL), threads(3THR), attributes(5)

454 man pages section 3: Networking Library Functions • Last Revised 1 Aug 2001

NAME rpc\_control – library routine for manipulating global RPC attributes for client and server applications

**SYNOPSIS** 

bool t rpc control(int op, void \*info);

DESCRIPTION This RPC library routine allows applications to set and modify global RPC attributes that apply to clients as well as servers. At present, it supports only server side operations. This function allows applications to set and modify global attributes that apply to client as well as server functions. op indicates the type of operation, and info is a pointer to the operation specific information. The supported values of *op* and their argument types, and what they do are:

RPC_SVC_MTMODE_SET	int *	set multithread mode
RPC_SVC_MTMODE_GET	int *	get multithread mode
RPC_SVC_THRMAX_SET	int *	set maximum number of threads
RPC_SVC_THRMAX_GET	int *	get maximum number of threads
RPC_SVC_THRTOTAL_GET	int *	get number of active threads
RPC_SVC_THRCREATES_GET	int *	get number of threads created
RPC_SVC_THRERRORS_GET	int *	get number of thread create errors
RPC_SVC_USE_POLLFD	int *	set number of file descriptors to unlimited
RPC_SVC_CONNMAXREC_SET	int *	set non-blocking max rec size
RPC_SVC_CONNMAXREC_GET	int *	get non-blocking max rec sizeThere are three
multithread (MT) modes.	These	are:

RPC SVC MT NONE Single threaded mode RPC SVC MT AUTO Automatic MT mode

(default)

RPC SVC MT USER User MT modeUnless the application sets the Automatic or User MT modes, it will stay in the default (single threaded) mode. See the *Network Interfaces Programmer's Guide* for the meanings of these modes and programming examples. Once a mode is set, it cannot be changed.

By default, the maximum number of threads that the server will create at any time is 16. This allows the service developer to put a bound on thread resources consumed by a server. If a server needs to process more than 16 client requests concurrently, the maximum number of threads must be set to the desired number. This parameter may be set at any time by the server.

Set and get operations will succeed even in modes where the operations don't apply. For example, you can set the maximum number of threads in any mode, even though it makes sense only for the Automatic MT mode. All of the get operations except RPC SVC MTMODE GET apply only to the Automatic MT mode, so values returned in other modes may be undefined.

By default, RPC servers are limited to a maximum of 1024 file descriptors or connections due to limitations in the historical interfaces svc fdset(3NSL) and svc\_getreqset(3NSL). Applications written to use the preferred interfaces of svc pollfd(3NSL) and svc getreg poll(3NSL) can use an unlimited number of file descriptors. Setting info to point to a non-zero integer and op to RPC SVC USE POLLFD removes the limitation.

rpc co	ntrol(3NSL)
rpc_co	

Connection oriented RPC transports read RPC requests in blocking mode by default.
Thus, they may be adversely affected by network delays and broken clients.
RPC_SVC_CONNMAXREC_SET enables non-blocking mode and establishes the
maximum record size (in bytes) for RPC requests; RPC responses are not affected.
Buffer space is allocated as needed up to the specified maximum, starting at the
maximum or RPC_MAXDATASIZE, whichever is smaller.

The value established by RPC\_SVC\_CONNMAXREC\_SET is used when a connection is created, and it remains in effect for that connection until it is closed. To change the value for existing connections on a per-connection basis, see svc control(3NSL).

RPC\_SVC\_CONNMAXREC\_GET retrieves the current maximum record size. A zero value means that no maximum is in effect, and that the connections are in blocking mode.

*info* is a pointer to an argument of type int. Non-connection RPC transports ignore RPC\_SVC\_CONNMAXREC\_SET and RPC\_SVC\_CONNMAXREC\_GET.

**RETURN VALUES** This routine returns TRUE if the operation was successful and returnsFALSE otherwise.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

## **SEE ALSO** rpcbind(1M), rpc(3NSL), rpc\_svc\_calls(3NSL), attributes(5)

Network Interfaces Programmer's Guide

rpc_gss_getcred – get credentials of client		
<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
<pre>bool_t rpc_gss_getcred(struct svc_req *req, rpc_gss_rawcred_ t</pre>		
rpc_gss_getcred() is used by a server to fetch the credentials of a client. These credentials may either be network credentials (in the form of a rpc_gss_rawcred_t structure) or UNIX credentials.		
For more information on RPCSEC_GSS data types, see the rpcsec_gss(3NSL) man page.		
Essentially, rpc_gss_getcred() passes a pointer to a request (svc_req) as well as pointers to two credential structures and a user-defined cookie; if rpc_gss_getcred() is successful, at least one credential structure is "filled out" with values, as is, optionally, the cookie.		
<i>req</i> Pointer to the received service request. svc_req is an RPC structure containing information on the context of an RPC invocation, such as program, version, and transport informatio		ormation on the context of an RPC
rcred	structure contains the ve protocol being used; the session (as strings); prin rpc_gss_principal_ the security service (inter-	s_rawcred_t structure pointer. This ersion number of the RPCSEC_GSS e security mechanism and QOPs for this ncipal names for the client (as a _t structure) and server (as a string); and egrity, privacy, etc., as an enum). If an sted in these values, it may pass NULL for
<i>ucred</i> The caller's UNIX credentials, in the form of a pointer to a pointer to a rpc_gss_ucred_t structure, which includes the client's uid and gids. If an application is not interested in these values, it may pass NULL for this parameter.		
<i>cookie</i> A four-byte quantity that an application may use in any manner it wants to; RPC does not interpret it. (For example, a cookie may be a pointer or index to a structure that represents a context initiator.) See also rpc_gss_set_callback(3NSL).		
<pre>rpc_gss_getcred() returns TRUE if it is successful; otherwise, use rpc_gss_get_error() to get the error associated with the failure.</pre>		
See attributes(	5) for descriptions of the	following attributes:
ATT	RIBUTE TYPE	ATTRIBUTE VALUE
MT-Level		MT-Safe
	<pre>#include <rpc bool_t="" rp="" rpc_gss<="" th=""><th><pre>#include <rpc rpcsec_gss.h=""> bool_t rpc_gss_getcred(struct svo</rpc></pre></th></rpc></pre>	<pre>#include <rpc rpcsec_gss.h=""> bool_t rpc_gss_getcred(struct svo</rpc></pre>

## rpc\_gss\_getcred(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Packages	SUNWrsg, SUNWrsgx

SEE ALSO rpc(3NSL), rpc\_gss\_set\_callback(3NSL), rpc\_gss\_set\_svc\_name(3NSL), rpcsec\_gss(3NSL), attributes(5)

ONC+ Developer's Guide

Network Working Group RFC 2078

NAME	rpc_gss_get_error – get error codes on failure		
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
	<pre>bool_t rpc_gss_get_error(rpc_gss_error_t*error);</pre>		
DESCRIPTION	rpc_gss_get_error() fetches an error code when an RPCSEC_GSS routine fails.		
	<pre>rpc_gss_get_error() uses a rpc_gss_error_t structure of the following form:</pre>		
	<pre>typedef struct {     int rpc_gss_error;</pre>		
	Currently the only error codes defined for t	his function are	
		* no error */ * system error */	
PARAMETERS	Information on RPCSEC_GSS data types for parameters may be found on the rpcsec_gss(3NSL) man page.		
	error A rpc_gss_error_t structure. If the rpc_gss_error field is equal to RPC_GSS_ER_SYSTEMERROR, the system_error field will be set to the value of errno.		
<b>RETURN VALUES</b>	Unless there is a failure indication from an invoked RPCSEC_GSS function, rpc_gss_get_error() does not set error to a meaningful value.		
ATTDIDITTCC	See attributes(5) for descriptions of the following attributes:		
ATTRIBUTES			
ATTRIDUTES	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
AI I KIDU I ES	ATTRIBUTE TYPE MT-Level	ATTRIBUTE VALUE MT-Safe	
AI I KIDU I ES			
SEE ALSO	MT-Level	MT-Safe SUNWrsg, SUNWrsgx	
	MT-Level Packages	MT-Safe SUNWrsg, SUNWrsgx	
	MT-Level Packages perror(3C), rpc(3NSL), rpcsec_gss(3N	MT-Safe SUNWrsg, SUNWrsgx	
	MT-Level Packages perror(3C), rpc(3NSL), rpcsec_gss(3N ONC+ Developer's Guide	MT-Safe SUNWrsg, SUNWrsgx	
SEE ALSO	MT-Level Packages perror(3C), rpc(3NSL), rpcsec_gss(3N ONC+ Developer's Guide Network Working Group RFC 2078	MT-Safe SUNWrsg, SUNWrsgx	
SEE ALSO	MT-Level Packages perror(3C), rpc(3NSL), rpcsec_gss(3N ONC+ Developer's Guide Network Working Group RFC 2078	MT-Safe SUNWrsg, SUNWrsgx	
SEE ALSO	MT-Level Packages perror(3C), rpc(3NSL), rpcsec_gss(3N ONC+ Developer's Guide Network Working Group RFC 2078	MT-Safe SUNWrsg, SUNWrsgx	

## rpc\_gss\_get\_mechanisms(3NSL)

NAME	rpc_gss_get_mechanisms, rpc_gss_get_mech_info, rpc_gss_get_versions, rpc_gss_is_installed – get information on mechanisms and RPC version		
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
	<pre>char **rpc_gss_get_mechanisms();</pre>		
	<pre>char **rpc_gss_get_mech_info(char *mech, rpc_gss_service_t</pre>		
	bool_t <b>rpc_gss</b>	<b>_get_versions</b> (u_:	<pre>int *vers_hi, u_int *vers_lo);</pre>
	bool_t rpc_gss	_is <b>installed</b> (cha	ar *mech);
DESCRIPTION	These "convenience and versions of RE		ormation on available security mechanisms
	rpc_gss_get_me	echanisms()	Returns a list of supported security mechanisms as a null-terminated list of character strings.
	rpc_gss_get_me	ech_info()	Takes two arguments: an ASCII string representing a mechanism type, for example, kerberosv5, and a pointer to a rpc_gss_service_t enum. rpc_gss_get_mech_info() will return NULL upon error or if no /etc/gss/qop file is present. Otherwise, it returns a null-terminated list of character strings of supported Quality of Protections (QOPs) for this mechanism. NULL or empty list implies only that the default QOP is available and can be specified to routines that need to take a QOP string parameter as NULL or as an empty string.
	rpc_gss_get_ve	ersions()	Returns the highest and lowest versions of RPCSEC_GSS supported.
	<pre>rpc_gss_is_installed()</pre>		Takes an ASCII string representing a mechanism, and returns TRUE if the mechanism is installed.
PARAMETERS	Information on RP rpcsec_gss(3NS		for parameters may be found on the
	mech		esenting the security mechanism in use. Valid ound in the /etc/gss/mech file.
	service		<pre>ss_service_t enum, representing the ce (privacy, integrity, or none).</pre>

rpc\_gss\_get\_mechanisms(3NSL)

FILES ATTRIBUTES	<pre>/etc/gss/mech File contain /etc/gss/qop File contain See attributes(5) for descriptions of the ATTRIBUTE TYPE MT-Level</pre>	ATTRIBUTE VALUE MT-Safe
	Availability	SUNWrsg (32–bit) SUNWrsgx (64–bit)
SEE ALSO	<pre>rpc(3NSL),rpcsec_gss(3NSL), mech(4), qop(4), attributes(5) ONC+ Developer's Guide Linn, J. RFC 2743, Generic Security Service Application Program Interface Version 2, Update 1. Network Working Group. January 2000.</pre>	
NOTES	This function will change in a future release.	

# rpc\_gss\_get\_principal\_name(3NSL)

NAME	rpc_gss_get_principal_name – Get principal names at server		
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
	<pre>bool_t rpc_gss_get_principal_name(rpc_gss_principal_ *principal,</pre>		
DESCRIPTION	Servers need to be able to operate on a client's principal name. Such a name is stored by the server as a rpc_gss_principal_t structure, an opaque byte string which can be used either directly in access control lists or as database indices which can be used to look up a UNIX credential. A server may, for example, need to compare a principal name it has received with the principal name of a known entity, and to do that, it must be able to generate rpc_gss_principal_t structures from known entities.		
	rpc_gss_get_principal_name() takes as input a security mechanism, a pointer to a rpc_gss_principal_t structure, and several parameters which uniquely identify an entity on a network: a user or service name, a node name, and a domain name. From these parameters it constructs a unique, mechanism-dependent principal name of the rpc_gss_principal_t structure type.		
PARAMETERS	How many of the identifying parameters ( <i>name</i> , <i>node</i> , and domain) are necessary to specify depends on the mechanism being used. For example, Kerberos V5 requires only a user name but can accept a node and domain name. An application can choose to set unneeded parameters to NULL.		
	Information on RPCSEC_GSS rpcsec_gss(3NSL) man pag	data types for parameters may be found on the ge.	
	principal	An opaque, mechanism-dependent structure representing the client's principal name.	
	mech	An ASCII string representing the security mechanism in use. Valid strings may be found in the /etc/gss/mech file, or by using rpc_gss_get_mechanisms().	
	name	A UNIX login name (for example, 'gwashington') or service name, such as 'nfs'.	
	node	A node in a domain; typically, this would be a machine name (for example, 'valleyforge').	
	domain	A security domain; for example, a DNS, NIS, or NIS+ domain name ('eng.company.com').	
<b>RETURN VALUES</b>		name () returns TRUE if it is successful; otherwise, use get the error associated with the failure.	
FILES	/etc/gss/mech	File containing valid security mechanisms	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		

rpc\_gss\_get\_principal\_name(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Packages	SUNWrsg, SUNWrsgx

## SEE ALSO free(3C), rpc(3NSL), rpc\_gss\_get\_mechanisms(3NSL), rpc\_gss\_set\_svc\_name(3NSL), rpcsec\_gss(3NSL), mech(4), attributes(5)

ONC+ Developer's Guide

Network Working Group RFC 2078

**NOTES** Principal names may be freed up by a call to free(3C). A principal name need only be freed in those instances where it was constructed by the application. (Values returned by other routines point to structures already existing in a context, and need not be freed.)

## rpc\_gss\_max\_data\_length(3NSL)

NAME	rpc_gss_max_data_length, rpc_gss_svc_max_data_length – get maximum data length for transmission		
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
	<pre>int rpc_gss_max_data_length(AUTH *handle, int max_tp_unit_len);</pre>		
	<pre>int rpc_gss_svc_max_data_length(struct svc_req *req, int</pre>		
DESCRIPTION	Performing a security transformation on a piece of data generally produces data with a different (usually greater) length. For some transports, such as UDP, there is a maximum length of data which can be sent out in one data unit. Applications need to know the maximum size a piece of data can be before it's transformed, so that the resulting data will still "fit" on the transport. These two functions return that maximum size.		
	<pre>rpc_gss_max_data_length() is the client-side version; rpc_gss_svc_max_data_length() is the server-side version.</pre>		
PARAMETERS	con rpo are	itext is cre c_gss_se	text handle of type AUTH, returned when a eated (for example, by eccreate(). Security service and QOP othis handle, eliminating any need to to
	•	e maximu nsport.	m size of a piece of data allowed by the
	info	ormation	an RPC svc_req structure, containing on the context (for example, program credentials).
<b>RETURN VALUES</b>	Both functions return the maximum size of untransformed data allowed, as an int.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	MT-Level		MT-Safe

SUNWrsg, SUNWrsgx

**SEE ALSO** rpc(3NSL), rpcsec\_gss(3NSL), attributes(5)

ONC+ Developer's Guide

Packages

Network Working Group RFC 2078

NAME	rpc_gss_mech_to_oid, rpc_gss_qop_to_num – map mechanism, QOP strings to non-string values		
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>		
	<pre>bool_t rpc_gss_mech_to_oid(charc*mech, rpc_gss_OIDc*oid);</pre>		
	<pre>bool_t rpc_gss_qop_to_num(char *qop, char *mech, u_int *num);</pre>		
DESCRIPTION	Because in-kernel RPC routines use non-string values for mechanism and Quality of Protection (QOP), these routines exist to map strings for these attributes to their non-string counterparts. (The non-string values for QOP and mechanism are also found in the /etc/gss/qop and /etc/gss/mech files, respectively.) rpc_gss_mech_to_oid() takes a string representing a mechanism, as well as a pointer to a rpc_gss_OID object identifier structure. It then gives this structure values corresponding to the indicated mechanism, so that the application can now use the OID directly with RPC routines. rpc_gss_qop_to_num() does much the same thing, taking strings for QOP and mechanism and returning a number.		
PARAMETERS	Information on RPCSEC_GSS data types for parameters may be found on the rpcsec_gss(3NSL) man page.		
	mech	An ASCII string representing the security mechanism in use. Valid strings may be found in the /etc/gss/mech file.	
	oid	An object identifier of type rpc_gss_OID, whose elements are usable by kernel-level RPC routines.	
	qop	This is an ASCII string which sets the quality of protection (QOP) for the session. Appropriate values for this string may be found in the file /etc/gss/qop.	
	num	The non-string value for the QOP.	
<b>RETURN VALUES</b>	Both functions retu	return TRUE if they are successful, FALSE otherwise.	
FILES	/etc/gss/mech	File containi	ng valid security mechanisms
	/etc/gss/qop	File containi	ng valid QOP values
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTR		ATTRIBUTE VALUE
	MT-Level		MT-Safe
	Packages		SUNWrsg, SUNWrsgx
SEE ALSO		gss_get_error(3NSL), L),mech(4),qop(4),att:	rpc_gss_get_mechanisms(3NSL), ributes(5)

ONC+ Developer's Guide

rpc\_gss\_mech\_to\_oid(3NSL)

Network Working Group RFC 2078

466 man pages section 3: Networking Library Functions • Last Revised 11 May 1998

		ipe_gss_seccicate(5105L)
NAME	rpc_gss_seccreate – create a se	ecurity context using the RPCSEC_GSS protocol
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>	
	<pre>AUTH *rpc_gss_seccreate(CLIENT *clnt, char *principal, char *mechanism, rpc_gss_service_t service_type, char *qop, rpc_gss_options_req_t *options_req, rpc_gss_options_ret_t *options_ret);</pre>	
DESCRIPTION	<pre>rpc_gss_seccreate() is used by an application to create a security context using the RPCSEC_GSS protocol, making use of the underlying GSS-API network layer. rpc_gss_seccreate() allows an application to specify the type of security mechanism (for example, Kerberos v5), the type of service (for example, integrity checking), and the Quality of Protection (QOP) desired for transferring data.</pre>	
PARAMETERS	Information on RPCSEC_GSS data types for parameters may be found on the rpcsec_gss(3NSL) man page.	
	clnt	This is the RPC client handle. <i>clnt</i> may be obtained, for example, from clnt_create().
	principal	This is the identity of the server principal, specified in the form <i>service@host</i> , where <i>service</i> is the name of the service the client wishes to access and <i>host</i> is the fully qualified name of the host where the service resides — for example, nfs@mymachine.eng.company.com.
	mechanism	This is an ASCII string which indicates which security mechanism to use with this data. Appropriate mechanisms may be found in the file /etc/gss/mech; additionally, rpc_gss_get_mechanisms() returns a list of supported security mechanisms (as null-terminated strings).
	service_type	This sets the initial type of service for the session — privacy, integrity, authentication, or none.
	qop	This is an ASCII string which sets the quality of protection (QOP) for the session. Appropriate values for this string may be found in the file /etc/gss/qop. Additionally, supported QOPs are returned (as null-terminated strings) by rpc_gss_get_mech_info().
	options_req	This structure contains options which are passed directly to the underlying GSS_API layer. If the caller specifies NULL for this parameter, defaults are used. (See NOTES, below.)
	options_ret	These GSS-API options are returned to the caller. If the caller does not need to see these options, then it may specify NULL for this parameter. (See NOTES, below.)

rpc\_gss\_seccreate(3NSL)

<b>RETURN VALUES</b>	rpc_gss_seccreate() returns a security context handle (an RPC authentication	
	handle) of type AUTH. If $rpc_gss_seccreate()$ cannot return successfully, the	
	application can get an error number by calling rpc_gss_get_error().	

	11 0	0 <u>1 _</u> <u>1 _</u> ()
FILES	/etc/gss/mech	File containing valid security mechanisms
	/etc/gss/qop	File containing valid QOP values .

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWrsg (32-bits)
	SUNWrsgx (64–bits)

SEE ALSO auth\_destroy(3NSL), rpc(3NSL), rpc\_gss\_get\_error(3NSL), rpc\_gss\_get\_mechanisms(3NSL), rpcsec\_gss(3NSL), mech(4), qop(4), attributes(5)

ONC+ Developer's Guide

Linn, J. RFC 2743, Generic Security Service Application Program Interface Version 2, Update 1. Network Working Group. January 2000.

**NOTES** Contexts may be destroyed normally, with auth\_destroy(). See auth\_destroy(3NSL)

NAME	rpc_gss_set_callba	ck – specify callback for context
SYNOPSIS	<pre>#include <rpc rpcsec_gss.h=""></rpc></pre>	
	<pre>bool_t rpc_gss_set_callback(struct rpc_gss_callback_t *cb);</pre>	
DESCRIPTION	first used. This use rpc_gss_set_ca	t to specify a callback routine so that it knows when a context gets er-defined callback may be specified through the allback() routine. The callback routine is invoked the first time a data exchanges, after the context is established for the specified ion.
	The user-defined o	allback routine should take the following form:
		truct svc_req *req, gss_cred_id_t deleg, gss_context, rpc_gss_lock_t *lock, void **cookie);
PARAMETERS	6       rpc_gss_set_callback() takes one argument: a pointer to a rpc_gss_callback_t structure. This structure contains the RPC program and version number as well as a pointer to a user-defined callback() routine. (For a description of rpc_gss_callback_t and other RPCSEC_GSS data types, see the rpcsec_gss(3NSL) man page.)         The user-defined callback() routine itself takes the following arguments:         req       Pointer to the received service request. svc_req is an RPC structure containing information on the context of an RPC invocation, such as program, version, and transport information.	
	deleg	Delegated credentials, if any. (See NOTES, below.)
	gss_context	GSS context (allows server to do GSS operations on the context to test for acceptance criteria). (See NOTES, below.)
	lock	This parameter is used to enforce a particular QOP and service for a session. This parameter points to a RPCSEC_GSS rpc_gss_lock_t structure. When the callback is invoked, the rpc_gss_lock_t.locked field is set to TRUE, thus locking the context. A locked context will reject all requests having different values for QOP or service than those specified by the raw_cred field of the rpc_gss_lock_t structure.
	cookie	A four-byte quantity that an application may use in any manner it wants to — RPC does not interpret it. (For example, the cookie could be a pointer or index to a structure that represents a context initiator.) The cookie is returned, along with the caller's credentials, with each invocation of rpc_gss_getcred().
RETURN VALUES	rpc_gss_set_ca otherwise.	allback() returns TRUE if the use of the context is accepted; false
	-	

## rpc\_gss\_set\_callback(3NSL)

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

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
	Packages	SUNWrsg, SUNWrsgx
SEE ALSO	rpc(3NSL), rpc_gss_getcred(3NSL), rp	ocsec_gss(3NSL), attributes(5)
	ONC+ Developer's Guide	
	Network Working Group RFC 2078	
NOTES	If a server does not specify a callback, all in	coming contexts will be accepted.
	Because the GSS-API is not currently expor mentioned for informational purposes only may choose to do nothing with them.	sed, the <i>deleg</i> and <i>gss_context</i> arguments are , and the user-defined callback function

## rpc\_gss\_set\_defaults(3NSL)

			1 -0
NAME	rpc_gss_set_defau	llts – change service, QOF	' for a session
SYNOPSIS	#include <rpc rp<="" th=""><th>csec_gss.h&gt;</th><th></th></rpc>	csec_gss.h>	
	bool_t <b>rpc_gss</b> char *qop)		*auth, rpc_gss_service_t service,
DESCRIPTION	integrity, authenti	cation, or none) and Qual	plication to change the service (privacy, lity of Protection (QOP) for a transfer e session (unless changed again).
PARAMETERS	Information on RPCSEC_GSS data types for parameters may be found on the rpcsec_gss(3NSL) man page.		
	auth	An RPC authentication rpc_gss_seccreate	
	service		gss_service_t, representing one of the rity service: authentication, privacy,
	дор	A string representing Q found in the file /etc/ rpc_gss_get_mech_:	
RETURN VALUES	<pre>rpc_gss_set_svc_name() returns TRUE if it is successful; otherwise, use rpc_gss_get_error() to get the error associated with the failure.</pre>		
FILES	/etc/gss/qop File containing valid QOPs		
ATTRIBUTES	See attributes(	(5) for descriptions of the	following attributes:
	ΔΤΤ	RIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level		MT-Safe
	Packages		SUNWrsg, SUNWrsgx
SEE ALSO	rpc(3NSL), rpc_ attributes(5) ONC+ Developer's Network Working C	Guide	NSL), rpcsec_gss(3NSL), qop(4),

Networking Library Functions 471

rpc\_gss\_set\_svc\_name(3NSL)

1 -0			
NAME	rpc_gss_set_svc_na	ame – send a principal na	ame to a server
SYNOPSIS	#include <rpc rpd<="" th=""><th>csec_gss.h&gt;</th><th></th></rpc>	csec_gss.h>	
		_ <b>set_svc_name</b> (char int <i>program</i> , u_int <i>v</i>	* <i>principal</i> , char * <i>mechanism</i> , u_int ersion);
DESCRIPTION	rpc_gss_set_svc_name() sets the name of a principal the server is to represent. If a server is going to act as more than one principal, this procedure can be invoked for every such principal.		
PARAMETERS	Information on RP rpcsec_gss(3NS		r parameters may be found on the
	principal	An ASCII string represe the form of <i>service@host</i>	enting the server's principal name, given in
	mech	An ASCII string represe strings may be found in rpc_gss_get_mechan	enting the security mechanism in use. Valid a the /etc/gss/mech file, or by using nisms().
	req_time	that the <i>req_time</i> is a hir	r which a credential should be valid. Note at to the underlying mechanism. The actual will remain valid is mechanism dependent. he actual time will be
	program	The RPC program num	ber for this service.
	version	The RPC version numb	er for this service.
RETURN VALUES		—	c if it is successful; otherwise, use ssociated with the failure.
FILES	/etc/gss/mech	File containi	ng valid security mechanisms
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level		MT-Safe
	Packages		SUNWrsg, SUNWrsgx
SEE ALSO	rpc(3NSL), rpc_g	gss_get_mechanisms( incipal_name(3NSL),	
	ONC+ Developer's (	Guide	

Linn, J., *RFC 2078, Generic Security Service Application Program Interface, Version 2,* Network Working Group, January 1997.

rpc\_rac(3RAC)

NAME	rpc_rac, rac_drop,	rac_poll, rac_recv, rac_send – r	emote asynchronous calls
SYNOPSIS	<b>cc</b> [ <i>flag</i> ] <i>file</i> #include <rpc rp<br="">#include <rpc ra<="" th=""><th></th><th> ]</th></rpc></rpc>		]
	void <b>rac_drop</b> (	CLIENT *cl, void *h);	
	enum clnt_stat	<pre>rac_poll(CLIENT *cl, voi</pre>	id *h);
	enum clnt_stat	<pre>rac_recv(CLIENT *cl, voi</pre>	id * <i>h</i> );
		CLIENT *cl, rpcproc_t pr proc_t xresults, void *result	<pre>oc, xdrproc_t xargs, void sp, struct timeval timeout);</pre>
DESCRIPTION	that allows message time between whe	ges to be sent using the RPC pro	a special interface to the RPC library otocol without blocking during the eply is received. To RPC servers, RAC es.
	A RAC message is allowing the client whether the return rac_recv() is us waiting for the return	sent using rac_send(). This to conduct other processing. W hed value from the call has beer red to collect the returned value	When the client wants to determine received, rac_poll() is used. e; it can also be used to block while op() is used to inform the RPC
	rac_drop()	message to the server is gener	ac_send() currently in progress. No cated by this call, but any subsequent will be silently dropped. It also frees
		After a call to rac_drop() th It may no longer be used in an	ne handle referred to by <i>h</i> is invalid. ny asynchronous operation.
	<pre>rac_poll()</pre>		us of the call currently in progress on nandle> tuple referred to by <i>cl</i> and <i>h</i> .
		rac_poll() return values an	re:
		RPC_SUCCESS	A reply has been received and is available for reading by rac_recv().
		RPC_INPROGRESS	No reply has been received. The call referred to by the given handle has not yet timed out.
		RPC_TIMEDOUT	No reply has been received. The call referred to by the given handle

rpc\_rac(3RAC)

n to by <i>h</i> is
to by <i>h</i> is
tly in JENT, Ile.
ssociated ndle is vhile cket.
ed to
er a is in has
RPC of y be used r a reply > tuple.
nous RPC a() call pends on ing s been
ons may
to by <i>h</i> is tly in JENT, de.
present UCCESS, g. In this s puld

			attempt to resend the call.
		XDR decode procedure call to rac_recv(), the	received, rac_recv() will invoke the specified in the rac_send() call. After a e handle referred to by <i>h</i> is invalid. It may y asynchronous operation.
	rac_send()	specified procedure. It c is the address of the pro in which to place the res used to encode and dece valid pointer when rac	sends to the server) an RPC call to the loes not await a reply from the server. <i>argsp</i> cedure's arguments, <i>resultsp</i> is the address sults, xargs and <i>xresults</i> are XDR functions ode respectively. Note: <i>resultsp</i> must be a _recv() is called. <i>timeout</i> should contain the application is willing to wait for a
		handle, is returned. This asynchronous calls to pe	e handle, known as the asynchronous s handle is to be used in subsequent oll for the status of the call (rac_poll()), ults of the call (rac_recv()), or cancel the
		On failure, ( <i>void</i> *) 0 is r	eturned.
		by calling clnt_geter failure (see rpc(3NSL)).	plication may retrieve the RPC failure code r() immediately after a rac_send() Possible errors include both transient port failures) and permanent ones (such as
			the same client handle are permitted, but table perturbations to the current timeout the RPC library.
		in progress before it is c	limit on the amount of time a call may be onsidered to have failed. This method was on the number of retries because of a ependence.
ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATTE	RIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level		Unsafe

# SEE ALSO poll(2), rpc(3NSL), rpc\_clnt\_create(3NSL), rpc\_clnt\_calls(3NSL), xdr(3NSL), attributes(5)

476 man pages section 3: Networking Library Functions • Last Revised 21 Jul 1998

WARNINGS The RAC interface is not the recommended interface for having multiple RPC requests outstanding. The preferred method of accomplishing this in the Solaris environment is to use synchronous RPC calls with threads. The RAC interface is provided as a service to developers interested in porting RPC applications to Solaris 2.0. Use of this interface will degrade the performance of normal synchronous RPC calls (see rpc clnt calls(3NSL)). For these reasons, use of this interface is disparaged.

The library librac must be linked before libns1 to use RAC. If the libraries are not linked in the correct order, then the results are indeterminate.

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

rpcsec\_gss(3NSL)

NAME	rpcsec_gss - security flavor incorporating GSS-API protections
SYNOPSIS	cc [ flag ] filelnsl [ library ] #include <rpc rpcsec_gss.h=""></rpc>
DESCRIPTION	RPCSEC_GSS is a security flavor which sits "on top" of the GSS-API (Generic Security Service API) for network transmissions. Applications using RPCSEC_GSS can take advantage of GSS-API security features; moreover, they can use any security mechanism (such as RSA public key or Kerberos) that works with the GSS-API.
	The GSS-API offers two security services beyond the traditional authentication services (AUTH_DH, AUTH_SYS, and AUTH_KERB): integrity and privacy. With integrity, the system uses cryptographic checksumming to ensure the authenticity of a message (authenticity of originator, recipient, and data); privacy provides additional security by encrypting data. Applications using RPCSEC_GSS specify which service they wish to use. Type of security service is mechanism-independent.
	Before exchanging data with a peer, an application must establish a context for the exchange. RPCSEC_GSS provides a single function for this purpose, rpc_gss_seccreate(), which allows the application to specify the security mechanism, Quality of Protection (QOP), and type of service at context creation. (The QOP parameter sets the cryptographic algorithms to be used with integrity or privacy, and is mechanism-dependent.) Once a context is established, applications can reset the QOP and type of service for each data unit exchanged, if desired.
	Valid mechanisms and QOPs may be obtained from configuration files or from the name service. Each mechanism has a default QOP.
	Contexts are destroyed with the usual RPC auth_destroy() call.
Data Structures	Some of the data structures used by the RPCSEC_GSS package are shown below.
	rpc_gss_service_t
	This enum defines the types of security services the context may have. rpc_gss_seccreate() takes this as one argument when setting the service type for a session.
	<pre>typedef enum {     rpc_gss_svc_default = 0,     rpc_gss_svc_none = 1,     rpc_gss_svc_integrity = 2,     rpc_gss_svc_privacy = 3 } rpc_gss_service_t ;</pre>
	rpc_gss_options_req_t
	Structure containing options passed directly through to the GSS-API. rpc_gss_seccreate() takes this as an argument when creating a context.
	<pre>typedef struct {     int req_flags; /*GSS request bits */</pre>

rpcsec\_gss(3NSL)

```
int time_req; /*requested credential lifetime */
gss_cred_id_t my_cred; /*GSS credential struct*/
gss_channel_bindings_t;
input_channel_bindings;
} rpc_gss_options_req_t ;
```

#### rpc\_gss\_OID

This data type is used by in-kernel RPC routines, and thus is mentioned here for informational purposes only.

typedef struct {
 u\_int length;
 void \*elements
} \*rpc\_gss\_OID;

#### rpc\_gss\_options\_ret\_t

Structure containing GSS-API options returned to the calling function, rpc\_gss\_seccreate().MAX\_GSS\_MECH is defined as 128.

```
typedef struct {
    int major_status;
    int minor_status;
    u_int rpcsec_version /*vers. of RPCSEC_GSS */
    int ret_flags
    int time_req
    gss_ctx_id_t gss_context;
    char actual_mechanism[MAX_GSS_MECH]; /*mechanism used*/
} rpc_gss_options_ret_t;
```

```
rpc_gss_principal_t
```

The (mechanism-dependent, opaque) client principal type. Used as an argument to the rpc\_gss\_get\_principal\_name() function, and in the gsscred table. Also referenced by the rpc\_gss\_rawcred\_t structure for raw credentials (see below).

```
typedef struct {
    int len;
    char name[1];
} *rpc_gss_principal_t;
```

```
rpc_gss_rawcred_t
```

Structure for raw credentials. Used by rpc\_gss\_getcred() and rpc\_gss\_set\_callback().

```
typedef struct {
    u_int version; /*RPC version # */
    char *mechanism; /*security mechanism*/
    char *qop; /*Quality of Protection*/
    rpc_gss_principal_t client_principal; /*client name*/
    char *svc_principal; /*server name*/
```

Networking Library Functions 479

```
rpcsec_gss(3NSL)
```

```
rpc_gss_service_t
                                           service;
                                                           /*service (integrity, etc.)*/
                    } rpc gss rawcred t;
                    rpc gss ucred t
                    Structure for UNIX credentials. Used by rpc gss getcred() as an alternative to
                    rpc gss rawcred t.
                    typedef struct {
                      uid_t uid; /*user ID*/
gid_t gid.
                       gid_t gid;
                                      /*group ID*/
                       short gidlen;
                       git_t *gidlist; /*list of groups*/
                    } rpc gss ucred t;
                    rpc gss callback t
                    Callback structure used by rpc gss set callback().
                    typedef struct {
                       u_int program; /*RPC program #*/
u_int version; /*RPC version #*/
                      u_int program;
                       bool_t (*callback)(); /*user-defined callback routine*/
                    } rpc gss callback t;
                    rpc gss lock t
                    Structure used by a callback routine to enforce a particular QOP and service for a
                    session. The locked field is normally set to FALSE; the server sets it to TRUE in order
                    to lock the session. (A locked context will reject all requests having different QOP and
                    service values than those found in the raw cred structure.) For more information, see
                    the rpc gss set callback(3NSL) man page.
                    typedef struct {
                       bool t
                                             locked;
                       rpc gss rawcred t
                                            *raw cred;
                    } rpc_gss_lock_t;
                    rpc gss error t
                    Structure used by rpc_gss_get_error() to fetch an error code when a
                    RPCSEC GSS routine fails.
                    typedef struct {
                       int rpc_gss_error;
                       int system_error; /*same as errno*/
                    } rpc_gss_error_t;
Index to Routines
                    The following lists RPCSEC GSS routines and the manual reference pages on which
                    they are described. An (S) indicates it is a server-side function:
```

```
Routine (Manual Page)
            Description
         rpc gss seccreate(3NSL)
            Create a secure RPCSEC GSS context
         rpc gss set defaults(3NSL)
            Switch service, QOP for a session
         rpc gss max data length(3NSL)
            Get maximum data length allowed by transport
         rpc gss set svc name(3NSL)
            Set server's principal name (S)
         rpc gss getcred(3NSL)
            Get credentials of caller (S)
         rpc gss set callback(3NSL)
            Specify callback to see context use (S)
         rpc gss get principal name(3NSL)
            Get client principal name (S)
         rpc gss svc max data length(3NSL)
            Get maximum data length allowed by transport (S)
         rpc gss get error(3NSL)
            Get error number
         rpc_gss_get_mechanisms(3NSL)
            Get valid mechanism strings
         rpc gss get mech info(3NSL)
            Get valid QOP strings, current service
         rpc_gss_get_versions(3NSL)
            Get supported RPCSEC GSS versions
         rpc gss is installed(3NSL)
            Checks if a mechanism is installed
         rpc gss mech to oid(3NSL)
            Maps ASCII mechanism to OID representation
         rpc gss qop to num(3NSL)
            Maps ASCII QOP, mechansim to u_int number
Utilities
         The gsscred utility manages the gsscred table, which contains mappings of
         principal names between network and local credentials. See gsscred(1M).
 FILES
         /etc/gss/mech
                                     List of installed mechanisms
         /etc/gss/qop
                                     List of valid QOPs
```

## rpcsec\_gss(3NSL)

## ATTRIBUTES

 $\mathbf{S}$  | See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWrsg (32-bit)
	SUNWrsgx (64-bit)

## SEE ALSO gsscred(1M), rpc(3NSL), rpc\_clnt\_auth(3NSL), xdr(3NSL), attributes(5), environ(5)

ONC+ Developer's Guide

Linn, J. RFC 2743, Generic Security Service Application Program Interface Version 2, Update 1. Network Working Group. January 2000.

NAME	rpc_soc, authdes_create, authunix_create, authunix_create_default, callrpc, clnt_broadcast, clntraw_create, clnttcp_create, clntudp_bufcreate, clntudp_create, get_myaddress, getrpcport, pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set, pmap_unset, registerrpc, svc_fds, svc_getcaller, svc_getreq, svc_register, svc_unregister, svcfd_create, svcraw_create, svctcp_create, svcudp_bufcreate, svcudp_create, xdr_authunix_parms – obsolete library routines for RPC
SYNOPSIS	#define PORTMAP #include <rpc rpc.h=""></rpc>
	<pre>AUTH *authdes_create(char *name, uint_t window, struct sockaddr_in     *syncaddr, des_block *ckey);</pre>
	AUTH <b>*authunix_create</b> (char <b>*</b> <i>host</i> , uid_t <i>uid</i> , gid_t <i>gid</i> , int <i>grouplen</i> , gid_t <b>*</b> <i>gidlistp</i> );
	AUTH <b>*authunix_create_default</b> void););
	<pre>callrpc(char *host, rpcprog_t prognum, rpcvers_t versnum, rpcproc_t</pre>
	<pre>enum clnt_stat_clnt_broadcast(rpcprog_t prognum, rpcvers_t     versnum, rpcproc_t procnum, xdrproc_t inproc, char *in, xdrproc_t     outproc, char *out, resultproc_teachresult);</pre>
	CLIENT *clntraw_create(rpcproc_t procnum, rpcvers_t versnum);
	CLIENT *clnttcp_create(struct sockaddr_in *addr, rpcprog_t prognum, rpcvers_t versnum, int *fdp, uint_t sendz, uint_t recvsz);
	<pre>CLIENT *clntudp_bufcreate(struct sockaddr_in *addr, rpcprog_t     prognum, rpcvers_t versnum, struct timeval wait, int *fdp, uint_t     sendz, uint_t recvsz);</pre>
	CLIENT <b>*clntudp_create</b> (struct sockaddr_in * <i>addr</i> , rpcprog_t <i>prognum</i> , struct timeval <i>wait</i> , int * <i>fdp</i> );
	<pre>void get_myaddress(struct sockaddr_in *addr);</pre>
	<pre>ushort getrpcport(char *host, rpcprog_t prognum, rpcvers_t versnum,</pre>
	<pre>struct pmaplist *pmap_getmaps(struct sockaddr_in *addr);</pre>
	<pre>ushort pmap_getport(struct sockaddr_in *addr, rpcprog_t prognum, rpcvers_t versnum, rpcprot_t protocol);</pre>
	<pre>enum clnt_stat pmap_rmtcall(struct sockaddr_in *addr, rpcprog_t     prognum, rpcvers_t versnum, rpcproc_t progcnum, caddr_t in,     xdrproct_t inproc, caddr_t out, cdrproct_t outproc, struct     timeval tout, rpcport_t *portp);</pre>
	<pre>bool_t pmap_set(rpcprog_t prognum, rpcvers_t versnum, rpcprot_t</pre>
	Networking Library Functions 483

```
rpc_soc(3NSL)
```

```
bool t pmap unset(rpcprog t prognum, rpcvers t versnum);
                 int svc fds;
                 struct sockaddr in *svc getcaller(SVCXPRT *xprt);
                 void svc getreq(int rdfds);
                 SVCXPRT *svcfd create(int fd, uint t sendsz, uint t recvsz);
                 SVCXPRT *svcraw createvoid););
                 SVCXPRT *svctcp create(int fd, uint t sendsz, uint t recvsz);
                 SVCXPRT *svcudp bufcreate(int fd, uint t sendsz, uint t recvsz);
                 SVCXPRT *svcudp create(int fd);
                 registerrpc (rpcprog t prognum, rpcvers t versnum, rpcproc t procnum,
                      char * (*procname) (), xdrproc t inproc, xdrproc t outproc);
                 bool tsvc register(SVCXPRT *xprt, rpcprog t prognum, rpcvers t
                      versnum, void (*dispatch(), int protocol);
                 void svc unregister(rpcprog t prognum, rpcvers t versnum);
                 bool t xdr authunix parms(XDR *xdrs, struct authunix parms *supp);
DESCRIPTION
                 RPC routines allow C 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. Finally, the procedure call returns to the client.
                 The routines described in this manual page have been superseded by other routines.
                 The preferred routine is given after the description of the routine. New programs
                 should use the preferred routines, as support for the older interfaces may be dropped
                 in future releases.
                                 File Descriptors
                                 Transport independent RPC uses TLI as its transport interface instead
                                 of sockets.
                                 Some of the routines described in this section (such as
                                 clnttcp create()) take a pointer to a file descriptor as one of the
                                 parameters. If the user wants the file descriptor to be a socket, then
                                 the application will have to be linked with both librpcsoc and
                                 libnsl. If the user passed RPC ANYSOCK as the file descriptor, and
                                 the application is linked with libnsl only, then the routine will
                                 return a TLI file descriptor and not a socket.
      Routines
                                 The following routines require that the header <rpc/rpc.h> be
                                 included. The symbol PORTMAP should be defined so that the
```

appropriate function declarations for the old interfaces are included through the header files.

authdes create()

authdes\_create() is the first of two routines which interface to the RPC secure authentication system, known as DES authentication. The second is authdes\_getucred(), below. Note: the keyserver daemon keyserv(1M) must be running for the DES authentication system to work.

authdes create(), used on the client side, returns an authentication handle that will enable the use of the secure authentication system. The first parameter *name* is the network name, or netname, of the owner of the server process. This field usually represents a hostname derived from the utility routine host2netname(), but could also represent a user name using user2netname() (see secure rpc(3NSL)). The second field is window on the validity of the client credential, given in seconds. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter syncaddr is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server's clock, and will not attempt resynchronizations. If an address is supplied, however, then the system will use the address for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter *ckey* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If it is supplied, however, then it will be used instead.

Warning: this routine exists for backward compatibility only, and is obsoleted by authdes\_seccreate() (see secure rpc(3NSL)).

authunix\_create()

Create and return an RPC authentication handle that contains .UX authentication information. The parameter *host* is the name of the machine on which the information was created; *uid* is the user's user ID; *gid* is the user's current group ID; *grouplen* and *gidlistp* refer to a counted array of groups to which the user belongs.

Warning: it is not very difficult to impersonate a user.

Warning: this routine exists for backward compatibility only, and is obsoleted by authsys\_create() (see rpc\_clnt\_auth(3NSL)).

authunix_create_default() Call authunix_create() with the appropriate parameters.
Warning: this routine exists for backward compatibility only, and is obsoleted by authsys_create_default() (see rpc_clnt_auth(3NSL)).
<pre>callrpc() 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, and out is the address of where to place the result(s). This routine returns 0 if it succeeds, or the value of enum clnt_stat cast to an integer if it fails. The routine clnt_perrno() (see rpc_clnt_calls(3NSL)) is handy for translating failure statuses into messages.</pre>
Warning: you do not have control of timeouts or authentication using this routine. This routine exists for backward compatibility only, and is obsoleted by rpc_call() (see rpc_clnt_calls(3NSL)).
<pre>clnt_stat_clnt_broadcast()   Like callrpc(), except the call message is broadcast to all locally   connected broadcast nets. Each time the caller receives a response,   this routine calls eachresult(), whose form is:</pre>
<pre>eachresult(char *out, struct sockaddr_in *addr);</pre>
where <i>out</i> is the same as <i>out</i> passed to clnt_broadcast(), except that the remote procedure's output is decoded there; <i>addr</i> points to the address of the machine that sent the results. If eachresult() returns 0. clnt_broadcast() waits for more replies; otherwise it returns with appropriate status. If eachresult() is NULL, clnt_broadcast() returns without waiting for any replies.
Broadcast packets are limited in size to the maximum transfer unit of the transports involved. For Ethernet, the callers argument size is approximately 1500 bytes. Since the call message is sent to all connected networks, it may potentially lead to broadcast storms. clnt_broadcast() uses SB AUTH_SYS credentials by default (see rpc_clnt_auth(3NSL)). This routine exists for backward compatibility only, and is obsoleted by rpc_broadcast() (see rpc_clnt_calls(3NSL)).

#### clntraw\_create()

This routine creates an internal, memory-based RPC client for the remote program *prognum*, version *versnum*. The transport used to pass messages to the service is actually a buffer within the process's address space, so the corresponding RPC server should live in the same address space; see svcraw\_create(). This allows simulation of RPC and acquisition of RPC overheads, such as round trip times, without any kernel interference. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only, and has the same functionality as clnt\_raw\_create() (see rpc clnt create(3NSL)), which obsoletes it.

#### clnttcp\_create()

This routine creates an RPC client for the remote program *prognum*, version *versnum*; the client uses TCP/IP as a transport. The remote program is located at Internet address *addr*. If *addr->sin\_port* is 0,, then it is set to the actual port that the remote program is listening on (the remote rpcbind service is consulted for this information). The parameter *\*fdp* is a file descriptor, which may be open and bound; if it is RPC\_ANYSOCK, then this routine opens a new one and sets *\*fdp*. Refer to the File Descriptor section for more information. Since TCP-based RPC uses buffered I/O, 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.

Warning: this routine exists for backward compatibility only. clnt\_create(), clnt\_tli\_create(), or clnt\_vc\_create() (see rpc\_clnt\_create(3NSL)) should be used instead.

#### clntudp\_bufcreate()

Create a client handle for the remote program *prognum*, on *versnum*; the client uses UDP/IP as the transport. The remote program is located at the Internet address *addr*. If *addr->sin\_port* is 0, it is set to port on which the remote program is listening on (the remote rpcbind service is consulted for this information). The parameter *\*fdp* is a file descriptor, which may be open and bound; if it is RPC\_ANYSOCK, then this routine opens a new one and sets *\*fdp*. Refer to the File Descriptor section for more information. The UDP transport resends the call message in intervals of wait time 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 rpc\_clnt\_calls(3NSL)). If successful it returns a client handle, otherwise it returns NULL. The error can

	<pre>be printed using the clnt_pcreateerror() (see rpc_clnt_create(3NSL)) routine.</pre>
	The user can specify the maximum packet size for sending and receiving by using <i>sendsz</i> and <i>recvsz</i> arguments for UDP-based RPC messages.
	Warning: if <i>addr-&gt;sin_port</i> is 0 and the requested version number <i>versnum</i> is not registered with the remote portmap service, it returns a handle if at least a version number for the given program number is registered. The version mismatch is discovered by a clnt_call() later (see rpc_clnt_calls(3NSL)).
	Warning: this routine exists for backward compatibility only. clnt_tli_create() or clnt_dg_create() (see rpc_clnt_create(3NSL)) should be used instead.
c	<pre>Intudp_create() This routine creates an RPC client handle for the remote program prognum, version versnum; the client uses UDP/IP as a transport. The remote program is located at Internet address addr. If addr-&gt;sin_port is 0, then it is set to actual port that the remote program is listening on (the remote rpcbind service is consulted for this information). The parameter *fdp is a file descriptor, which may be open and bound; if it is RPC_ANYSOCK, then this routine opens a new one and sets *fdp. Refer to the File Descriptor section for more information. The UDP transport resends the call message in intervals of wait time 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 rpc_clnt_calls(3NSL)). clntudp_create() returns a client handle on success, otherwise it returns NULL. The error can be printed using the clnt_pcreateerror() (see rpc_clnt_create(3NSL)) routine.</pre>
	Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.
	Warning: this routine exists for backward compatibility only. clnt_create(), clnt_tli_create(), or clnt_dg_create() (see rpc_clnt_create(3NSL)) should be used instead.
g	<pre>et_myaddress() Places the local system's IP address into *addr, without consulting the library routines that deal with /etc/hosts. The port number is always set to htons(PMAPPORT).</pre>

Warning: this routine is only intended for use with the RPC library. It returns the local system's address in a form compatible with the RPC library, and should not be taken as the system's actual IP address. In fact, the *\*addr* buffer's host address part is actually zeroed. This address may have only local significance and should NOT be assumed to be an address that can be used to connect to the local system by remote systems or processes.

Warning: this routine remains for backward compatibility only. The routine netdir\_getbyname() (see netdir(3NSL)) should be used with the name HOST\_SELF to retrieve the local system's network address as a *netbuf* structure.

getrpcport()

getrpcport() returns the port number for the version versnum
of the RPC program prognum running on host and using protocol
proto.getrpcport() returns 0 if the RPC system failed to contact
the remote portmap service, the program associated with prognum
is not registered, or there is no mapping between the program and
a port.

Warning: This routine exists for backward compatibility only. Enhanced functionality is provided by rpcb\_getaddr() (see rpcbind(3NSL)).

pmaplist()

A user interface to the portmap service, which returns a list of the current RPC program-to-port mappings on the host located at IP address *addr*. This routine can return NULL. The command 'rpcinfo -p' uses this routine.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpcb\_getmaps() (see rpcbind(3NSL)).

#### pmap\_getport()

A user interface to the portmap service, which returns the port number on which waits a service that supports program *prognum*, version *versnum*, and speaks the transport protocol associated with *protocol*. The value of *protocol* is most likely IPPROTO\_UDP or IPPROTO\_TCP. A return value of 0 means that the mapping does not exist or that the RPC system failured to contact the remote portmap service. In the latter case, the global variable rpc createerr contains the RPC status.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpcb\_getaddr() (see rpcbind(3NSL)).

ľ	<pre>pmap_rmtcall() Request that the portmap on the host at IP address *addr make an RPC on the behalf of the caller to a procedure on that host. *portp is modified to the program's port number if the procedure succeeds. The definitions of other parameters are discussed in callrpc() and clnt_call() (see rpc_clnt_calls(3NSL)).</pre>
	Note: this procedure is only available for the UDP transport.
	Warning: if the requested remote procedure is not registered with the remote portmap then no error response is returned and the call times out. Also, no authentication is done.
	Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpcb_rmtcall() (see rpcbind(3NSL)).
I	<pre>pmap_set() A user interface to the portmap service, that establishes a mapping between the triple [prognum, versnum, protocol] and port on the machine's portmap service. The value of protocol may be IPPROTO_UDP or IPPROTO_TCP. Formerly, the routine failed if the requested port was found to be in use. Now, the routine only fails if it finds that port is still bound. If port is not bound, the routine completes the requested registration. This routine returns 1 if it succeeds, 0 otherwise. Automatically done by svc_register(). Warning: this routine exists for backward compatibility only,</pre>
	enhanced functionality is provided by rpcb_set() (see rpcbind(3NSL)).
Ĭ	<pre>pmap_unset() A user interface to the portmap service, which destroys all mapping between the triple [prognum, versnum, all-protocols] and port on the machine's portmap service. This routine returns one if it succeeds, 0 otherwise.</pre>
	Warning: this routine exists for backward compatibility only, enhanced functionality is provided by rpcb_unset() (see rpcbind(3NSL)).
5	<pre>svc_fds() A global variable reflecting the RPC service side's read file descriptor bit mask; it is suitable as a parameter to the select() call. This is only of interest if a service implementor does not call svc_run(), but rather does his own asynchronous event processing. This variable is read-only (do not pass its address to</pre>

select()!), yet it may change after calls to svc\_getreq() or any creation routines. Similar to svc\_fdset, but limited to 32 descriptors.

Warning: this interface is obsoleted by svc\_fdset (see rpc svc calls(3NSL)).

svc\_getcaller()

This routine returns the network address, represented as a struct sockaddr\_in, of the caller of a procedure associated with the RPC service transport handle, *xprt*.

Warning: this routine exists for backward compatibility only, and is obsolete. The preferred interface is svc\_getrpccaller() (see rpc\_svc\_reg(3NSL)), which returns the address as a struct netbuf.

#### svc\_getreq()

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 the select() call has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced. This routine is similar to svc\_getreqset() but is limited to 32 descriptors.

Warning: this interface is obsoleted by svc getreqset().

#### svcfd\_create()

Create a service on top of any open and bound descriptor. Typically, this descriptor is a connected file descriptor for a stream protocol. Refer to the File Descriptor section for more information. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, a reasonable default is chosen.

Warning: this interface is obsoleted by svc\_fd\_create() (see rpc svc create(3NSL)).

#### svcraw\_create()

This routine creates an internal, memory-based RPC service transport, to which it returns a pointer. 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 clntraw\_create(). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel interference. This routine returns NULL if it fails.

rpc svc create(3NSL)), which obsoletes it. svctcp create() This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor fd, which may be RPC ANYSOCK, in which case a new file descriptor is created. If the file descriptor is not bound to a local TCP port, then this routine binds it to an arbitrary port. Refer to the File Descriptor section for more information. Upon completion, *xprt*->xp fd is the transport's file descriptor, and *xprt->xp* port is the transport's port number. This routine returns NULL if it fails. Since TCP-based RPC uses buffered I/O, users may specify the size of buffers; values of 0 choose suitable defaults. Warning: this routine exists for backward compatibility only. svc create(), svc tli create(), or svc vc create() (see rpc svc create(3NSL)) should be used instead. svcudp bufcreate() This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor fd. If fd is RPC ANYSOCK then a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt*xp fd is the transport's file descriptor, and *xprt*->xp port is the transport's port number. Refer to the File Descriptor section for more information. This routine returns NULL if it fails. The user specifies the maximum packet size for sending and receiving UDP-based RPC messages by using the sendsz and recvsz parameters. Warning: this routine exists for backward compatibility only. svc tli create(), or svc dg create() (see rpc svc create(3NSL)) should be used instead. svcudp create() This routine creates a UDP/IP-based RPC service transport, to

Warning: this routine exists for backward compatibility only, and

has the same functionality of svc raw create() (see

which it returns a pointer. The transport is associated with the file descriptor *fd*, which may be RPC\_ANYSOCK, in which case a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt*->xp\_fd is the transport's file descriptor, and *xprt*->xp\_port is the transport's port number. This routine returns NULL if it fails.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

Warning: this routine exists for backward compatibility only. svc\_create(), svc\_tli\_create(), or svc\_dg\_create() (see rpc\_svc\_create(3NSL)) should be used instead.

registerrpc()

Register program *prognum*, procedure *procname*, and version *versnum* with the RPC service package. If a request arrives for program *prognum*, version *versnum*, and procedure *procnum*, *procname* is called with a pointer to its parameter(s); *procname* should return a pointer to its static result(s); *inproc* is used to decode the parameters while *outproc* is used to encode the results. This routine returns 0 if the registration succeeded, –1 otherwise.

svc\_run() must be called after all the services are registered.

Warning: this routine exists for backward compatibility only, and is obsoleted by rpc reg().

svc\_register()

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *protocol* is 0, the service is not registered with the portmap service. If *protocol* is non-zero, then a mapping of the triple [*prognum*, *versnum*, *protocol*] to *xprt*->xp\_port is established with the local portmap service (generally *protocol* is 0, IPPROTO\_UDP or IPPROTO\_TCP). The procedure *dispatch* has the following form:

dispatch(struct svc\_req \*request, SVCXPRT \*xprt);

The svc\_register() routine returns one if it succeeds, and 0 otherwise.

Warning: this routine exists for backward compatibility only; enhanced functionality is provided by svc reg().

#### svc\_unregister()

Remove all mapping of the double [*prognum*, *versnum*] to dispatch routines, and of the triple [*prognum*, *versnum*, *all-protocols*] to port number from portmap.

Warning: this routine exists for backward compatibility, enhanced functionality is provided by svc\_unreg().

rpc_soc(3NSL)			
		Used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the	
		Warning: this routine exists for backward compatibility only, and is obsoleted by xdr_authsys_parms() (see rpc_xdr(3NSL)).	
ATTRIBUTES	See attributes(5) for description	See attributes(5) for descriptions of the following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Unsafe	
SEE ALSO	netdir_getbyname(3NSL), rpc rpc_clnt_calls(3NSL), rpc_cl rpc_svc_calls(3NSL), rpc_svc rpc_svc_err(3NSL), rpc_svc_n rpcbind(3NSL), secure_rpc(3N	<pre>keyserv(1M), rpcbind(1M), rpcinfo(1M), netdir(3NSL), netdir_getbyname(3NSL), rpc (3NSL), rpc_clnt_auth(3NSL), rpc_clnt_calls(3NSL), rpc_clnt_create(3NSL), rpc_svc_calls(3NSL), rpc_svc_create(3NSL), rpc_svc_err(3NSL), rpc_svc_reg(3NSL), rpc_xdr(3NSL), rpcbind(3NSL), secure rpc(3NSL), select(3C),</pre>	
	xdr_authsys_parms(3NSL), lik librpcsoc(3LIBUCB), attribut		
NOTES	These interfaces are unsafe in mult	These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.	
	1		

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	These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.	
	These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as svc_run()) are called when the server is initiated.	
	In the current implementation, the service transport handle SVCXPRT contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the Automatic or User MT modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. 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.	
Routines	<pre>#include <rpc rpc.h=""></rpc></pre>	
<pre>int svc_dg_enablecache(SVCXPRT *xprt, const uint_t cache_size); This function allocates a duplicate request cache for the service endpoint xpi enough to hold cache_size entries. Once enabled, there is no way to disable c This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</pre>		
	This function is safe in MT applications.	
	<pre>int svc_done(SVCXPRT *xprt); This function frees resources allocated to service a client request directed to the service endpoint xprt. This call pertains only to servers executing in the User M mode. In the User MT mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After svc_done() is invoked, the service endpoint xprt should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the rpc_control() call.</pre>	
	This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.	
	void svc_exit(void); This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes svc_run() to return.	
	If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the rpc_svc_create(3NSL) functions, or using xprt_register(3NSL).	

rpc\_svc\_calls(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 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 no 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 \*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 \*rdfds);

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; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

struct netbuf \*svc\_getrpccaller(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. It 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.

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.

## rpc\_svc\_calls(3NSL)

 $\label{eq:attributes} \textbf{ATTRIBUTES} ~|~ \textbf{See attributes}(5) ~ for descriptions of the following attributes:$ 

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	See NOTES below.
		·]
SEE ALSO	<pre>rpcgen(1), poll(2), rpc(3NSL), rpc_con rpc_svc_err(3NSL), rpc_svc_reg(3NSL) attributes(5)</pre>	
NOTES	<pre>svc_dg_enablecache() and svc_getrpccaller() are safe in multithreaded applications. svc_freeargs(), svc_getargs(), and svc_sendreply() are safe in MT applications utilizing the Automatic or User MT modes. svc_getreq_common(), svc_getreqset(), and svc_getreq_poll() are unsafe in multithreaded applications and should be called only from the main thread.</pre>	

NAME	svc_raw_create, svc_tli_create	svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, raw_create, svc_tli_create, svc_tp_create, svc_vc_create, svc_door_create – library nes for the creation of server handles	
SYNOPSIS	<pre>#include <rpc rpc.h=""></rpc></pre>		
	bool_t svc_control(SVC	<pre>XPRT *svc, const uint_t req, void *info);</pre>	
	<pre>int svc_create(const void (*dispatch)const struct svc_req *, const SVCXPRT *, const rpcprog_t prognum, const rpcvers_t versnum, const char *nettype);</pre>		
	<pre>void svc_destroy(SVCXPRT *xprt);</pre>		
	<pre>SVCXPRT *svc_dg_create(const int fildes, const uint_t sendsz, const uint_t recvsz);</pre>		
	<pre>SVCXPRT *svc_fd_create     uint_t recvsz);</pre>	<pre>(const int fildes, const uint_t sendsz, const</pre>	
	<pre>SVCXPRT *svc_raw_createvoid););</pre>		
	<pre>SVCXPRT *svc_tli_create(const int fildes, const struct netconfig *netconf, const struct t_bind *bind_addr, const uint_t sendsz, const uint_t recvsz);</pre>		
	*, const SVCXPRT *	<pre>(const void (*dispatch)const struct svc_req ), const rpcprog_t prognum, const rpcvers_t ct netconfig *netconf);</pre>	
	<pre>SVCXPRT *svc_vc_create     uint_t recvsz);</pre>	<pre>(const int fildes, const uint_t sendsz, const</pre>	
	<pre>SVCXPRT *svc_door_create(void (*dispatch)(struct svc_req *, SVCXPRT  *), const rpcprog_t prognum, const rpcvers_t versnum, const     uint_t sendsz);</pre>		
DESCRIPTION	procedure calls on servers acr	routines are part of the RPC library which allows C language programs to make dure calls on servers across the network. These routines deal with the creation o e handles. Once the handle is created, the server can be invoked by calling cun().	
Routines	See rpc(3NSL) for the definition of the SVCXPRT data structure.		
	<pre>svc_control()</pre>	A function to change or retrieve 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:	
		SVCGET_VERSQUIET If a request is received for a program number served by this server but the version number is outside the range registered with the server, an	

RPC\_PROGVERSMISMATCH error will normally be returned. *info* should be a pointer to an integer. Upon successful completion of the SVCGET\_VERSQUIET request, *\*info* contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an RPC\_PROGVERSMISMATCH error will be returned. 1 indicates that the out of range request will be silently ignored.

#### SVCSET VERSQUIET

If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC\_PROGVERSMISMATCH error will normally be returned. It is sometimes desirable to change this behavior. *info* should be a pointer to an integer which is either 0, indicating normal server behavior and an RPC\_PROGVERSMISMATCH 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 and connectionless transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program 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)

#### SVCSET RECVERRHANDLER

Attaches or detaches a disconnection handler to the service handle, *svc*, that will be called when a transport error arrives during the reception of a request or when the server is waiting for a request and the connection shuts down. This handler is only useful for a connection oriented service handle.

## rpc\_svc\_create(3NSL)

	<i>*info</i> contains the address of the error handler to attach, or NULL to detach a previously defined one. The error handler has two arguments. It has a pointer to the erroneous service handle. It also has an integer that indicates if the full service is closed (when equal to zero), or that only one connection on this service is closed (when not equal to zero).
	<pre>void handler (const SVCXPRT *svc, const bool_t isAConnection);</pre>
	With the service handle address, <i>svc</i> , the error handler is able to detect which connection has failed and to begin an error recovery process. The error handler can be called by multiple threads and should be implemented in an MT-safe way.
	SVCGET_RECVERRHANDLER Upon successful completion of the SVCGET_RECVERRHANDLER request, <i>*info</i> contains the address of the handler for receiving errors. Upon failure, <i>*info</i> contains NULL.
	This routine returns TRUE if the operation was successful. Otherwise, it returns false.
<pre>svc_create()</pre>	<pre>svc_create() creates server handles for all the transports belonging to the class nettype.</pre>
	<i>nettype</i> 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 <i>nettype</i> is NULL, it defaults to netpath.
	<pre>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.</pre>
<pre>svc_destroy()</pre>	A function macro that destroys the RPC service handle <i>xprt</i> . Destruction usually involves deallocation of private data structures, including <i>xprt</i> itself. Use of <i>xprt</i> is undefined after calling this routine.
<pre>svc_dg_create()</pre>	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. <i>sendsz</i> and <i>recvsz</i> are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor <i>fildes</i> 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.
<pre>svc_fd_create()</pre>	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. <i>sendsz</i> and <i>recvsz</i> indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns NULL if it fails, and an error message is logged.
<pre>svc_raw_create()</pre>	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: <pre>svc_run() should not be called when the raw interface is being used.</pre>
<pre>svc_tli_create()</pre>	This routine creates an RPC server handle, and returns a pointer to it. <i>fildes</i> is the file descriptor on which the service is listening. If <i>fildes</i> is RPC_ANYFD, it opens a file descriptor on the transport specified by <i>netconf</i> . If the file descriptor is unbound and <i>bindaddr</i> is non-null <i>fildes</i> is bound to the address specified by <i>bindaddr</i> , otherwise <i>fildes</i> 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 <i>sendsz</i> and <i>recvsz</i> ; 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.

	<pre>svc_tp_create()</pre>	network spe the rpcbin remote proc <i>versnum;</i> this svc_tp_cr	<pre>eate() creates a server handle for the ecified by netconf, and registers itself with d service. dispatch is called when there is a edure call for the given prognum and s requires calling svc_run(). eate() returns the service handle if it herwise a NULL is returned and an error ogged.</pre>
	<pre>svc_vc_create()</pre>	and returns it fails, and a specify the s parameters suitable defa	creates a connection-oriented RPC service a pointer to it. This routine returns NULL if an error message is logged. The users may size of the send and receive buffers with the <i>sendsz</i> and <i>recvsz</i> ; values of 0 choose aults. The file descriptor <i>fildes</i> should be bund. The server is not registered with the <i>M</i> ) service.
ATTRIBUTES	<pre>svc_door_create() See attributes(5) for descri</pre>	and returns mechanism processes or The user ma parameter so default buffo svc_door_ handle. Other rpc_creat rpcbind(1)	
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	Architecture		All
	Availability		SUNWcsl (32–bit)
			SUNWcslx (64–bit)
	Interface Stability		Evolving
	MT-Level		MT-Safe

SEE ALSO rpcbind(1M), rpc(3NSL), rpc\_clnt\_create(3NSL), rpc\_svc\_calls(3NSL), rpc\_svc\_err(3NSL), rpc\_svc\_reg(3NSL), attributes(5)

Networking Library Functions 503

rpc_	_svc_	_err(3NSL)
------	-------	------------

NAME	rpc_svc_err, svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprog, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors		
DESCRIPTION	These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.		
	These routines can be called by the server side dispatch function if there is any error in the transaction with the client.		
Routines	See rpc(3NSL) for the definition of the SVC	XPRT data structure.	
	<pre>#include <rpc rpc.h=""></rpc></pre>		
	<pre>void svcerr_auth(const SVCXPRT *xprt, const enum auth_stat why); Called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.</pre>		
	<pre>void svcerr_decode(const SVCXPRT *xprt); Called by a service dispatch routine that cannot successfully decode the remote parameters (see svc_getargs() in rpc_svc_reg(3NSL)).</pre>		
	<pre>void svcerr_noproc(const SVCXPRT *<i>xprt</i>); Called by a service dispatch routine that does not implement the procedure number that the caller requests.</pre>		
	<pre>void svcerr_noprog(const SVCXPRT *xprt); Called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.</pre>		
	<pre>void svcerr_progvers(const SVCXPRT *xprt, const rpcvers_t low_vers, const rpcvers_t high_vers); Called when the desired version of a program is not registered with the RPC package. low_vers is the lowest version number, and high_vers is the highest version number. Service implementors usually do not need this routine.</pre>		
	<pre>void svcerr_systemerr(const SVCXPRT *xprt); Called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.</pre>		
	<pre>void svcerr_weakauth(const SVCXPRT *xprt); Called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters. The routine calls svcerr_auth(xprt, AUTH_TOOWEAK).</pre>		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	

rpc\_svc\_err(3NSL)

SEE ALSO | rpc(3NSL), rpc\_svc\_calls(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_reg(3NSL), attributes(5)

# rpc\_svc\_input(3NSL)

NAME	rpc_svc_input, svc_add_input, svc_remove_input – declare or remove a callback on a file descriptor	
SYNOPSIS	<pre>#include <rpc rpc.h=""></rpc></pre>	
	<pre>typedef void (*svc_callback_t) (svc_input_id_t id, int fd, unsigned</pre>	
	<pre>svc_input_id_t svc_add_input(int fd, unsigned int revents,</pre>	
	<pre>void svc_remove_input(sv</pre>	c_input_t <i>id</i> );
DESCRIPTION	The following RPC routines are descriptor.	used to declare or remove a callback on a file
Routines	See rpc(3NSL) for the definition	of the SVCXPRT data structure.
	<pre>svc_add_input() This function is used to register a <i>callback</i> function on a file descriptor, <i>fd</i>. The file descriptor, <i>fd</i>, is the first parameter to be passed to svc_add_input(). This <i>callback</i> function will be automatically called if any of the events specified in the <i>events</i> parameter occur on this descriptor. The <i>events</i> parameter is used to specify when the callback is invoked. This parameter is a mask of poll events to which the user wants to listen. See poll(2) for further details of the events that can be specified.</pre>	
	The callback to be invoked is specified using the <i>callback</i> parameter. The <i>cookie</i> parameter can be used to pass any data to the <i>callback</i> function. This parameter is a user-defined value which is passed as an argument to the <i>callback</i> function, and it is not used by the Sun RPC library itself.	
	Several callbacks can be registered on the same file descriptor as long as each callback registration specifies a separate set of event flags.	
	The <i>callback</i> function is called with the registration <i>id</i> , the <i>fd</i> file descriptor, an <i>revents</i> value, which is a bitmask of all events concerning the file descriptor, and the <i>cookie</i> user-defined value.	
	Upon successful completion, the function returns a unique identifier for this registration, that can be used later to remove this callback. Upon failure, -1 is returned and errno is set to indicate the error.	
	The svc_add_input() func	tion will fail if:
	EINVAL	The <i>fd</i> or <i>events</i> parameters are invalid.
	EEXIST	A callback is already registered to the file descriptor with one of the specified events.
	ENOMEM Memory is exhausted.	

svc\_remove\_input()
This function is used to unregister a callback function on a file descriptor, fd. The id
parameter specifies the registration to be removed.

Upon successful completion, the function returns zero. Upon failure, -1 is returned and errno is set to indicate the error.

The svc\_remove\_input() function will fail if:

EINVAL The *id* parameter is invalid.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	All
Availability	SUNWcsl (32-bit)
	SUNWcslx (64-bit)
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

poll(2), rpc(3NSL), attributes (5)

rpc_svc_reg(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 rpcbind() (see rpcbind(1M)), and associate the given program and version number with the dispatch function. When the RPC server receives a RPC request, the library invokes the dispatch routine with the appropriate arguments.
Routines	See rpc(3NSL) for the definition of the SVCXPRT data structure.
	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>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); Register program prognum, procedure procname, and version versnum with the RPC service package. If a request arrives for program prognum, version versnum, and procedure procnum, procname is called with a pointer to its parameter(s); procname should return a pointer to its static result(s). The arg parameter to procname is a pointer to the (decoded) procedure argument. inproc is the XDR function used to decode the parameters while outproc is the XDR function used to encode the results. Procedures are registered on all available transports of the class nettype. See rpc(3NSL). This routine returns 0 if the registration succeeded, -1 otherwise. int svc_reg(const SVCXPRT *xprt, const rpcprog_t prognum, const rpcvers_t versnum, const void (*dispatch)(), const struct netconfig *netconf); Associates prognum and versnum with the service dispatch procedure, dispatch. If netconf is NULL, the service is not registered with the rpcbind service. For example, if a service has already been registered using some other means, such as inetd (see inetd(1M)), it will not need to be registered again. If netconf is non-zero, then a mapping of the triple [prognum, versnum, netconf-&gt;] to xprt-&gt; xp_ltaddr is established with the local rpcbind service.</pre>
	The svc_reg() routine returns 1 if it succeeds, and 0 otherwise.
	<pre>void svc_unreg(const rpcprog_t prognum, const rpcvers_t versnum); Remove from the rpcbind service, all mappings of the triple [prognum, versnum, all-transports] to network address and all mappings within the RPC service package of the double [prognum, versnum] to dispatch routines.</pre>
	int svc_auth_reg(const int <i>cred_flavor</i> , const enum auth_stat (*handler)()); Registers the service authentication routine <i>handler</i> with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type <i>cred_flavor</i> . 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 <i>cred_flavor</i> , the <i>handler</i> 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 <i>cred_flavor</i> already has an authentication handler registered for it, and $-1$ otherwise.		
	<pre>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.</pre>		
	<pre>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.</pre>		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level	MT-Safe	

SEE ALSO inetd(1M), rpcbind(1M), rpc(3NSL), rpc\_svc\_calls(3NSL), rpc\_svc\_create(3NSL), rpc\_svc\_err(3NSL), rpcbind(3NSL), select(3C), attributes(5) rpc\_xdr(3NSL)

NAME	rpc_xdr, xdr_accepted_reply, xdr_authsys_parms, xdr_callhdr, xdr_callmsg, xdr_opaque_auth, xdr_rejected_reply, xdr_replymsg – XDR library routines for remote procedure calls
SYNOPSIS	<pre>bool_t xdr_accepted_reply(XDR *xdrs, const struct accepted_reply</pre>
	<pre>bool_t xdr_authsys_parms(XDR *xdrs, struct authsys_parms *aupp);</pre>
	<pre>void xdr_callhdr(XDR *xdrs, struct rpc_msg *chdr);</pre>
	<pre>bool_t xdr_callmsg(XDR *xdrs, struct rpc_msg *cmsg);</pre>
	<pre>bool_t xdr_opaque_auth(XDR *xdrs, struct opaque_auth *ap);</pre>
	<pre>bool_t xdr_rejected_reply(XDR *xdrs, const struct rejected_reply</pre>
	<pre>bool_t xdr_replymsg(XDR *xdrs, const struct rpc_msg *rmsg);</pre>
DESCRIPTION	These routines are used for describing the RPC messages in XDR language. They should normally be used by those who do not want to use the RPC package directly. These routines return TRUE if they succeed, FALSE otherwise.
Routines	See rpc(3NSL) for the definition of the XDR data structure.
	<pre>#include <rpc rpc.h=""></rpc></pre>
	<pre>xdr_accepted_reply() Used to translate between RPC reply messages and their external representation. It includes the status of the RPC call in the XDR language format. In the case of success, it also includes the call results.</pre>
	xdr_authsys_parms() Used for describing UNIX operating system credentials. It includes machine-name, uid, gid list, etc.
	xdr_callhdr() Used for describing RPC call header messages. It encodes the static part of the call message header in the XDR language format. It includes information such as transaction ID, RPC version number, program and version number.
	<pre>xdr_callmsg() Used for describing RPC call messages. This includes all the RPC call information such as transaction ID, RPC version number, program number, version number, authentication information, etc. This is normally used by servers to determine information about the client RPC call.</pre>
	xdr_opaque_auth() Used for describing RPC opaque authentication information messages.

		ssages. It encodes the rejected RPC message in ssage could be rejected either because of version uthentication errors.
		ssages. It translates between the RPC reply ation. This reply could be either an acceptance,
TTRIBUTES	See attributes(5) for descriptions of	of the following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	Safe

# rstat(3RPC)

tat(3RPC)			
NAME	rstat, havedisk – get performance data from	a remote kernel	
SYNOPSIS	<pre>cc [ flag ] filelrpcsvc [ library ] #include <rpc rpc.h=""> #include <rpcsvc rstat.h=""></rpcsvc></rpc></pre>		
	<pre>enum clnt_stat rstat(char *host, struct statstime *statp);</pre>		
	<pre>int havedisk(char *host);</pre>		
PROTOCOL	/usr/include/rpcsvc/rstat.x		
DESCRIPTION	These routines require that the rpc.rstatd(1M) daemon be configured and available on the remote system indicated by <i>host</i> . The rstat() protocol is used to gather statistics from remote kernel. Statistics will be available on items such as paging, swapping, and cpu utilization.		
	<pre>rstat() fills in the statstime structure s allocated statstime structure.rstat() r otherwise a enum clnt_stat is returned w clnt_perrno(3NSL).</pre>	eturns RPC_SUCCESS if it was successful;	
	havedisk() returns 1 if <i>host</i> has disk, 0 if determined.	it does not, and -1 if this cannot be	
	The following XDR routines are available ir	librpcsvc:	
	xdr_statstime xdr_statsvar		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	rup(1), rpc.rstatd(1M), rpc_clnt_cal	ls(3NSL), attributes(5)	

512 man pages section 3: Networking Library Functions • Last Revised 30 Dec 1996

## rusers(3RPC)

NAME	rusers, rnusers – return information about u	users on remote machines	
SYNOPSIS	<pre>cc [ flag ] filelrpcsvc [ library ] #include <rpc rpc.h=""> #include <rpcsvc rusers.h=""></rpcsvc></rpc></pre>		
	enum clnt_stat <b>rusers</b> (char * <i>host</i> ,	<pre>struct utmpidlearr *up);</pre>	
	<pre>int rnusers(char *host);</pre>		
PROTOCOL	/usr/include/rpcsvc/rusers.x		
DESCRIPTION	These routines require that the rpc.rusersd(1M) daemon be configured and available on the remote system indicated by <i>host</i> . The rusers() protocol is used to retrieve information about users logged in on the remote system.		
	rusers() fills the utmpidlearr structure with data about <i>host</i> , and returns 0 if successful. <i>up</i> must point to an allocated utmpidlearr structure. If rusers() returns successful it will have allocated data structures within the <i>up</i> structure, which should be freed with xdr free(3NSL) when you no longer need them:		
	<pre>xdr_free(xdr_utimpidlearr, up);</pre>		
	On error, the returned value can be interpreted as an enum clnt_stat and can be displayed with clnt_perror(3NSL) or clnt_sperrno(3NSL).		
	See the header <rpcsvc rusers.h=""> for a</rpcsvc>	definition of struct utmpidlearr.	
	rnusers() returns the number of users logged on to <i>host</i> (-1 if it cannot determine that number).		
	The following XDR routines are available in librpcsvc:		
	xdr_utmpidlearr		
ATTRIBUTES	See attributes (5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	<pre>rusers(1), rpc.rusersd(1M), rpc_clnt attributes(5)</pre>	_calls(3NSL),xdr_free(3NSL),	
	I		

# rwall(3RPC)

NAME	rwall – write to specified remote machines		
SYNOPSIS	<pre>cc [ flag ] filelrpcsvc [ library ] #include <rpc rpc.h=""> #include <rpcsvc rwall.h=""></rpcsvc></rpc></pre>		
	<pre>enum clnt_stat rwall(char *host, c</pre>	har * <i>msg</i> );	
PROTOCOL	/usr/include/rpcsvc/rwall.x		
DESCRIPTION	These routines require that the rpc.rwalld(1M) daemon be configured and available on the remote system indicated by <i>host</i> .		
	<pre>rwall() executes wall(1M) on host. The rpc.rwalld process on host prints msg to all users logged on to that system.rwall() returns RPC_SUCCESS if it was successful; otherwise a enum clnt_stat is returned which can be displayed using clnt_perrno(3NSL).</pre>		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_calls(3NSL), attributes(5)</pre>		
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL),attributes(5)	
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL),attributes(5)	
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL),attributes(5)	
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL),attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL), attributes(5)	
SEE ALSO	rpc.rwalld(1M),wall(1M),rpc_clnt_	calls(3NSL), attributes(5)	
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL), attributes(5)	
SEE ALSO	<pre>rpc.rwalld(1M), wall(1M), rpc_clnt_</pre>	calls(3NSL), attributes(5)	

NAME	key_decryptsession, key_encryptsession, key_gendes, key_setsecret, key_secretkey_is_set, netname2host, netname2user, user2netname – library routines for secure remote procedure calls		
SYNOPSIS	<pre>int authdes_getucred(const struct authdes_cred *adc, uid_t *uidp, gid_t *gidp, short *gidlenp, gid_t *gidlist);</pre>		
	<pre>AUTH *authdes_seccreate(const char *name, const uint_t window,</pre>		
	<pre>int getnetname(char name [MAXNETNAMELEN+1]);</pre>		
	<pre>int host2netname(char name [MAXNETNAMELEN+1], const char *host,</pre>		
	<pre>int key_decryptsession(const char *remotename, des_block *deskey);</pre>		
	<pre>int key_encryptsession(const char *remotename, des_block *deskey);</pre>		
	<pre>int key_gendes(des_block *deskey);</pre>		
	<pre>int key_setsecret(const char *key);</pre>		
	<pre>int key_secretkey_is_setvoid););</pre>		
	<pre>int netname2host(const char *name, char *host, const int hostlen);</pre>		
	<pre>int netname2user(const char *name, uid_t *uidp, gid_t *gidp, int *gidlenp, gid_t *gidlist [NGRPS]);</pre>		
	<pre>int user2netname(char name [MAXNETNAMELEN+1], const uid_t uid,</pre>		
DESCRIPTION	RPC library routines allow C programs to make procedure calls on other machines across the network.		
	RPC supports various authentication flavors. Among them are:		
	AUTH_NONE No authentication (none).		
	AUTH_SYS Traditional UNIX-style authentication.		
	AUTH_DES DES encryption-based authentication.		
	The authdes_getucred() and authdes_seccreate() routines implement the AUTH_DES authentication flavor. The keyserver daemon keyserv (see keyserv(1M)) must be running for the AUTH_DES authentication system to work, and keylogin(1) must have been run. Only the AUTH_DES style of authentication is discussed here. For information about the AUTH_NONE and AUTH_SYS styles of authentication, refer to rpc_clnt_auth(3NSL).		
	The routines documented on this page are MT-Safe. See theman pages for the other authentication styles for their MT-level.		

#### secure\_rpc(3NSL)

Routines

es | See rpc(3NSL) for the definition of the AUTH data structure.

#include <rpc/rpc.h>
#include <sys/types.h>

authdes getucred()

authdes\_getucred() is the first of the two routines which interface to the RPC secure authentication system known as AUTH\_DES. The second is authdes\_seccreate(), below. authdes\_getucred() is used on the server side for converting an AUTH\_DES credential, which is operating system independent, into an AUTH\_SYS credential. This routine returns 1 if it succeeds, 0 if it fails.

*\*uidp* is set to the user's numerical ID associated with *adc. \*gidp* is set to the numerical ID of the user's group. *\*gidlist* contains the numerical IDs of the other groups to which the user belongs. *\*gidlenp* is set to the number of valid group ID entries in *\*gidlist* (see netname2user(), below).

Warning: authdes\_getucred() will fail if the authdes\_cred structure was created with the netname of a host. In such a case, netname2host() should be used on the host netname in the authdes\_cred structure to get the host name.

```
authdes seccreate()
```

authdes\_seccreate(), the second of two AUTH\_DES authentication routines, is used on the client side to return an authentication handle that will enable the use of the secure authentication system. The first parameter *name* is the network name, or *netname*, of the owner of the server process. This field usually represents a hostname derived from the utility routine host2netname(), but could also represent a user name using user2netname(), described below.

The second field is *window* on the validity of the client credential, given in seconds. If the difference in time between the client's clock and the server's clock exceeds *window*, the server will reject the client's credentials, and the clock will have to be resynchronized. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift.

The third parameter, *timehost*, the host's name, is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the *timehost* clock, and will not attempt resynchronizations. If a timehost is supplied, however, then the system will consult with the remote time service whenever resynchronization is required. This parameter is usually the name of the host on which the server is running.

The final parameter *ckey* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If *ckey* is supplied, then it will be used instead.

If authdes\_seccreate() fails, it returns NULL.

#### getnetname()

getnetname() returns the unique, operating system independent netname of the caller in the fixed-length array *name*. Returns 1 if it succeeds, and 0 if it fails.

#### host2netname()

Convert from a domain-specific hostname *host* to an operating system independent netname. Returns 1 if it succeeds, and 0 if it fails. Inverse of netname2host(). If *domain* is NULL, host2netname() uses the default domain name of the machine. If *host* is NULL, it defaults to that machine itself. If *domain* is NULL and *host* is a NIS name like "host1.ssi.sun.com," host2netname() uses the domain "ssi.sun.com" rather than the default domain name of the machine.

#### key decryptsession()

key\_decryptsession() is an interface to the keyserver daemon, which is associated with RPC's secure authentication system (AUTH\_DES authentication). User programs rarely need to call it, or its associated routines key encryptsession(), key gendes(), and key setsecret().

key\_decryptsession() takes a server netname *remotename* and a DES key *deskey*, and decrypts the key by using the the public key of the the server and the secret key associated with the effective UID of the calling process. It is the inverse of key\_encryptsession().

### key\_encryptsession()

key\_encryptsession() is a keyserver interface routine. It takes a server netname *remotename* and a DES key *deskey*, and encrypts it using the public key of the the server and the secret key associated with the effective UID of the calling process. It is the inverse of key\_decryptsession(). This routine returns 0 if it succeeds, -1 if it fails.

### key\_gendes()

key\_gendes () is a keyserver interface routine. It is used to ask the keyserver for a secure conversation key. Choosing one at random is usually not good enough, because the common ways of choosing random numbers, such as using the current time, are very easy to guess. This routine returns 0 if it succeeds, -1 if it fails.

### key\_setsecret()

key\_setsecret() is a keyserver interface routine. It is used to set the key for the effective UID of the calling process. This routine returns 0 if it succeeds, -1 if it fails.

### key\_secretkey\_is\_set()

key\_secretkey\_is\_set() is a keyserver interface routine that may be used to determine whether a key has been set for the effective UID of the calling process. If the keyserver has a key stored for the effective UID of the calling process, this routine returns 1. Otherwise it returns 0.

secure_rpc(3NSL)		
	<pre>netname2host() Convert from an operating system independent netname name to a domain-specific hostname host. hostlen is the maximum size of host. Returns 1 if it succeeds, and 0 if it fails. Inverse of host2netname().</pre>	
	<pre>netname2user() Convert from an operating system independent netname to a domain-specific user ID. Returns 1 if it succeeds, and 0 if it fails. Inverse of user2netname().</pre>	
	* <i>uidp</i> is set to the user's numerical ID associated with <i>name</i> . * <i>gidp</i> is set to the numerical ID of the user's group. <i>gidlist</i> contains the numerical IDs of the other groups to which the user belongs. * <i>gidlenp</i> is set to the number of valid group ID entries in <i>gidlist</i> .	
	user2netname() Convert from a domain-specific usernam netname. Returns 1 if it succeeds, and 0	
ATTRIBUTES	See $\texttt{attributes}(5)$ for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT-Level	MT-Safe
	<pre>chkey(1), keylogin(1), keyserv(1M), newkey(1M), rpc(3NSL), rpc_clnt_auth(3NSL), attributes(5)</pre>	
SEE ALSO		wkey(1M), rpc(3NSL),

NAME	send, sendto, sendmsg – send a message from a socket		
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl [ library ] #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>		
	<pre>ssize_t send(int s, const void *msg, size_t len, int flags);</pre>		
	<pre>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	<pre>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. s is a socket created with socket(3SOCKET).</pre>		
	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.		
	A return value of –1 indicates the message was not delivered	locally detected errors only. It does not implicitly mean d.	
	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 (see fcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.		
	The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:		
	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 and 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.	
	See recv(3SOCKET) for a description of the msghdr structure.		
<b>RETURN VALUES</b>	These calls return the number of bytes sent, or $-1$ if an error occurred.		
ERRORS	The calls fail if:		
	EBADF	<i>s</i> is an invalid file descriptor.	
	EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.	

Networking Library Functions 519

send(3SOCKET)

,			
	EINVAL	<i>tolen</i> is not the address fam	he size of a valid address for the specified ily.
	EMSGSIZE		equires that message be sent atomically, sage was too long.
	ENOMEM	There was ir the operation	nsufficient memory available to complete n.
	ENOSR		nsufficient STREAMS resources available ation to complete.
	ENOTSOCK	s is not a soc	ket.
	EWOULDBLOCK	The socket is operation w	s marked non-blocking and the requested ould block.
TTRIBUTES	See attributes(5) for descr	riptions of the	following attributes:
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	MT-Level		Safe
SEE ALSO	recv(3SOCKET), select(30		OCKET), getsockopt(3SOCKET), OCKET), attributes(5),
SEE ALSO			
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		
SEE ALSO	recv(3SOCKET), select(30		

NAME	send – send a mes	sage on a socket	
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	ssize_t <b>send</b> (i	int <i>socket</i> , const void * <i>bu</i>	ffer, size_t length, int flags);
DESCRIPTION	socket	Specifies the socket file descr	iptor.
	buffer	Points to the buffer containing	g the message to send.
	length	Specifies the length of the me	essage in bytes.
	flags	Specifies the type of message argument are formed by logi following flags:	e transmission. Values of this cally OR'ing zero or more of the
		MSG_EOR	Terminates a record (if supported by the protocol)
		MSG_OOB	Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.
	its peer. The send	() function sends a message of he peer of a connectionless soc	message from the specified socket to only when the socket is connected ket has been set via
			by the <i>length</i> argument. If the message col, send() fails and no data is
		tion of a call to send() does r –1 indicates only locally-detect	not guarantee delivery of the message. red errors.
	and the socket file space is available. be transmitted and	descriptor does not have O_N If space is not available at the d the socket file descriptor doe ect(3C) and poll(2) functions	hold the message to be transmitted IONBLOCK set, send() blocks until sending socket to hold the message to s have O_NONBLOCK set, send() can be used to determine when it is
	The socket in use send() function.	may require the process to hav	e appropriate privileges to use the
USAGE		tion is identical to sendto(3XN write() if no flags are used.	NET) with a null pointer <i>dest_len</i>

# send(3XNET)

RETURN VALUES	Upon successful completion, send() returns the number of bytes sent. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The send() function will fail if:		
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.	
	EBADF	The socket argument is not a valid file descriptor.	
	ECONNRESET	A connection was forcibly closed by a peer.	
	EDESTADDRREQ	The socket is not connection-mode and no peer address is set.	
	EFAULT	The <i>buffer</i> parameter can not be accessed.	
	EINTR	A signal interrupted send() before any data was transmitted.	
	EMSGSIZE	The message is too large be sent all at once, as the socket requires.	
	ENOTCONN	The socket is not connected or otherwise has not had the peer prespecified.	
	ENOTSOCK	The <i>socket</i> argument does not refer to a socket.	
	EOPNOTSUPP	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .	
	EPIPE	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process.	
	The send() function may fail	l if:	
	EACCES	The calling process does not have the appropriate privileges.	
	EIO	An I/O error occurred while reading from or writing to the file system.	
	ENETDOWN	The local interface used to reach the destination is down.	
	ENETUNREACH	No route to the network is present.	
	ENOBUFS	Insufficient resources were available in the system to perform the operation.	
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.	
	•		

### send(3XNET)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO connect(3XNET), getsockopt(3XNET), poll(2), recv(3XNET), recvfrom(3XNET), recvmsg(3XNET), select(3C), sendmsg(3XNET), sendto(3XNET), setsockopt(3XNET), shutdown(3XNET), socket(3XNET), attributes(5)

sendmsg(3XNET)			
NAME	sendmsg – send a	message on a socket using a m	essage structure
SYNOPSIS	<pre>cc [ flag ] fil #include <sys pre="" sc<=""></sys></pre>	elxnet [ <i>library</i> ] cket.h>	
	ssize_t <b>sendma</b>	<b>sg</b> (int <i>socket</i> , const struc	t msghdr * <i>message</i> , int <i>flags</i> );
DESCRIPTION	connectionless-mo sent to the address	unction sends a message throug ode socket. If the socket is conne s specified by <i>msghdr</i> . If the soc ss in <i>msghdr</i> is ignored.	ectionless-mode, the message will be
	The function takes	the following arguments:	
	socket	Specifies the socket file descri	ptor.
	message	address and the buffers for th	containing both the destination e outgoing message. The length and on the address family of the socket. gnored.
	flags	Specifies the type of message specify 0 or the following flag	transmission. The application may ;:
		MSG_EOR	Terminates a record (if supported by the protocol)
		MSG_OOB	Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific.
	the data to be sent set to the dimension storage area and the zero. The data from	. <i>msg_iov</i> points to an array of a on of this array. In each iovec the <i>iov_len</i> field gives its size in 1 m each storage area indicated b	cify zero or more buffers containing iovec structures; <i>msg_iovlen</i> must be structure, the <i>iov_base</i> field specifies a bytes. Some of these sizes can be y <i>msg_iov</i> is sent in turn. es not guarantee delivery of the
		value of –1 indicates only local	
	and the socket file blocks until space	descriptor does not have O_No is available. If space is not avai transmitted and the socket file	old the message to be transmitted ONBLOCK set, sendmsg() function lable at the sending socket to hold descriptor does have O_NONBLOCK
		cket protocol, sendmsg() will	specified address is a broadcast fail if the SO_BROADCAST option is

	The socket in use may require sendmsg() function.	e the process to have appropriate privileges to use the
USAGE	The select(3C) and poll(2) to send more data.	) functions can be used to determine when it is possible
<b>RETURN VALUES</b>		sendmsg() function returns the number of bytes sent. l errno is set to indicate the error.
ERRORS	The sendmsg() function wil	l fail if:
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.
	EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
	EBADF	The socket argument is not a valid file descriptor.
	ECONNRESET	A connection was forcibly closed by a peer.
	EFAULT	The <i>message</i> parameter, or storage pointed to by the <i>msg_name</i> , <i>msg_control</i> or <i>msg_iov</i> fields of the <i>message</i> parameter, or storage pointed to by the <i>iovec</i> structures pointed to by the <i>msg_iov</i> field can not be accessed.
	EINTR	A signal interrupted sendmsg() before any data was transmitted.
	EINVAL	The sum of the iov_len values overflows an ssize_t.
	EMSGSIZE	The message is to large to be sent all at once (as the socket requires), or the msg_iovlen member of the msghdr structure pointed to by <i>message</i> is less than or equal to 0 or is greater than IOV_MAX.
	ENOTCONN	The socket is connection-mode but is not connected.
	ENOTSOCK	The <i>socket</i> argument does not refer a socket.
	EOPNOTSUPP	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
	EPIPE	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process.
	If the address family of the so	ocket is AF_UNIX, then $sendmsg()$ will fail if:

sendmsg(3XNET)

unisg(SANET)		
	EIO	An I/O error occurred while reading from or writing to the file system.
	ELOOP	Too many symbolic links were encountered in translating the pathname in the socket address.
	ENAMETOOLONG	A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
	ENOENT	A component of the pathname does not name an existing file or the pathname is an empty string.
	ENOTDIR	A component of the path prefix of the pathname in the socket address is not a directory.
	The sendmsg() function may	/ fail if:
	EACCES	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
	EDESTADDRREQ	The socket is not connection-mode and does not have its peer address set, and no destination address was specified.
	EHOSTUNREACH	The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
	EIO	An I/O error occurred while reading from or writing to the file system.
	EISCONN	A destination address was specified and the socket is already connected.
	ENETDOWN	The local interface used to reach the destination is down.
	ENETUNREACH	No route to the network is present.
	ENOBUFS	Insufficient resources were available in the system to perform the operation.
	ENOMEM	Insufficient memory was available to fulfill the request.
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.
	If the address family of the so	cket is AF_UNIX, then sendmsg() may fail if:
	ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ATTRIBUTES	See attributes(5) for descri	ptions of the following attributes:

### sendmsg(3XNET)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO

poll(2) getsockopt(3XNET), recv(3XNET), recvfrom(3XNET), recvmsg(3XNET), select(3C), send(3XNET), sendto(3XNET), setsockopt(3XNET), shutdown(3XNET), socket(3XNET), attributes(5)

# sendto(3XNET)

NAME	sendto – send a m	essage on a socket	
SYNOPSIS	<b>cc</b> [ <i>flag</i> ] <i>file</i> #include <sys so<="" th=""><th>elxnet [ <i>libra</i> cket.h&gt;</th><th><i>rry</i> ]</th></sys>	elxnet [ <i>libra</i> cket.h>	<i>rry</i> ]
			st void *message, size_t length, int flags, st_addr, socklen_t dest_len);
DESCRIPTION	The sendto() function sends a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message will be sent to the address specified by <i>dest_addr</i> . If the socket is connection-mode, <i>dest_addr</i> is ignored.		
	The function takes	the following argu	ments:
	socket	Specifies the socke	t file descriptor.
	message	Points to a buffer of	containing the message to be sent.
	length	Specifies the size of	of the message in bytes.
	flags	1 71	of message transmission. Values of this ned by logically OR'ing zero or more of the
		MSG_EOR	Terminates a record (if supported by the protocol)
		MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
	dest_addr		ddr structure containing the destination address. mat of the address depend on the address et.
	dest_len	Specifies the lengt dest_addr argument	h of the sockaddr structure pointed to by the t.
		cket protocol, sendt	ast and the specified address is a broadcast co() will fail if the SO_BROADCAST option is
	The <i>dest_addr</i> argu specifies the lengtl	-	ddress of the target. The <i>length</i> argument
			dto() does not guarantee delivery of the sonly locally-detected errors.
			socket to hold the message to be transmitted have O_NONBLOCK set, sendto() blocks

		ce is not available at the sending socket to hold the d the socket file descriptor does have O_NONBLOCK
	The socket in use may require sendto() function.	e the process to have appropriate privileges to use the
USAGE	The select(3C) and poll(2) to send more data.	) functions can be used to determine when it is possible
<b>RETURN VALUES</b>	Upon successful completion, –1 is returned and errno is s	sendto() returns the number of bytes sent. Otherwise, et to indicate the error.
ERRORS	The sendto() function will	fail if:
	EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
	EAGAIN EWOULDBLOCK	The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.
	EBADF	The <i>socket</i> argument is not a valid file descriptor.
	ECONNRESET	A connection was forcibly closed by a peer.
	EFAULT	The <i>message</i> or <i>destaddr</i> parameter can not be accessed.
	EINTR	A signal interrupted sendto() before any data was transmitted.
	EMSGSIZE	The message is too large to be sent all at once, as the socket requires.
	ENOTCONN	The socket is connection-mode but is not connected.
	ENOTSOCK	The <i>socket</i> argument does not refer to a socket.
	EOPNOTSUPP	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
	EPIPE	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process.
	If the address family of the so	ocket is AF_UNIX, then sendto() will fail if:
	EIO	An I/O error occurred while reading from or writing to the file system.
	ELOOP	Too many symbolic links were encountered in translating the pathname in the socket address.

sendto(3XNET)

	ENAMETOOLONG	A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
	ENOENT	A component of the pathname does not name an existing file or the pathname is an empty string.
	ENOTDIR	A component of the path prefix of the pathname in the socket address is not a directory.
	The sendto() function may	fail if:
	EACCES	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
	EDESTADDRREQ	The socket is not connection-mode and does not have its peer address set, and no destination address was specified.
	EHOSTUNREACH	The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
	EINVAL	The <i>dest_len</i> argument is not a valid length for the address family.
	EIO	An I/O error occurred while reading from or writing to the file system.
	EISCONN	A destination address was specified and the socket is already connected.
	ENETDOWN	The local interface used to reach the destination is down.
	ENETUNREACH	No route to the network is present.
	ENOBUFS	Insufficient resources were available in the system to perform the operation.
	ENOMEM	Insufficient memory was available to fulfill the request.
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.
	If the address family of the so	cket is AF_UNIX, then sendto() may fail if:
	ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:

530 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

### sendto(3XNET)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO poll(2), getsockopt(3XNET), recv(3XNET), recvfrom(3XNET), recvmsg(3XNET), select(3C), send(3XNET), sendmsg(3XNET), setsockopt(3XNET), shutdown(3XNET), socket(3XNET), attributes(5)

setsockopt(3XNET)			
NAME	setsockopt – set the socket options		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	<pre>int setsockopt(int socket, int level, int option_name, const void*option_value, socklen_t option_len);</pre>		
DESCRIPTION	The setsockopt() function sets the option specified by the <i>option_name</i> argument, at the protocol level specified by the <i>level</i> argument, to the value pointed to by the <i>option_value</i> argument for the socket associated with the file descriptor specified by the <i>socket</i> argument.		
	The <i>level</i> argument specifies the protocol level at which the option resides. To set options at the socket level, specify the <i>level</i> argument as SOL_SOCKET. To set options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP (Transport Control Protocol), set <i>level</i> to the protocol number of TCP, as defined in the <netinet in.h=""> header, or as determined by using getprotobyname(3XNET).</netinet>		
	The <i>option_name</i> argument specifies a single option to set. The <i>option_name</i> argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretations. The <sys socket.h=""> header defines the socket level options. The options are as follows:</sys>		
	SO_DEBUG Turns on recording of debugging information. This option enables or disables debugging in the underlyin protocol modules. This option takes an int value. This a boolean option.		
	SO_BROADCAST	Permits sending of broadcast messages, if this is supported by the protocol. This option takes an int value. This is a boolean option.	
	SO_REUSEADDR	Specifies that the rules used in validating addresses supplied to bind(3XNET) should allow reuse of local addresses, if this is supported by the protocol. This option takes an int value. This is a boolean option.	
	SO_KEEPALIVE	Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an int value.	
		If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a SIGPIPE signal.	
		This is a boolean option.	
	I		

# setsockopt(3XNET)

	SO_LINGER	Lingers on a close(2) if data is present. This option controls the action taken when unsent messages queue on a socket and close(2) is performed. If SO_LINGER is set, the system blocks the process during close(2) until it can transmit the data or until the time expires. If SO_LINGER is not specified, and close(2) is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a linger structure, as defined in the <sys socket.h=""> header, to specify the state of the option and linger interval.</sys>	
	SO_OOBINLINE	Leaves received out-of-band data (data marked urgent) in line. This option takes an int value. This is a boolean option.	
	SO_SNDBUF	Sets send buffer size. This option takes an int value.	
	SO_RCVBUF	Sets receive buffer size. This option takes an int value.	
	SO_DONTROUTE	Requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an int value. This is a boolean option.	
	For boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.		
	Options at other protocol leve	ls vary in format and name.	
USAGE	The setsockopt() function provides an application program with the means to control socket behavior. An application program can use setsockopt() to allocate buffer space, control timeouts, or permit socket data broadcasts. The <sys socket.h=""> header defines the socket-level options available to setsockopt().</sys>		
	Options may exist at multiple protocol levels. The SO_ options are always present the uppermost socket level.		
RETURN VALUES	Upon successful completion, setsockopt() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The setsockopt() function	will fail if:	
	EBADF	The <i>socket</i> argument is not a valid file descriptor.	
	EDOM	The send and receive timeout values are too big to fit into the timeout fields in the socket structure.	

setsockopt(3XNET)

	EFAULT	The <i>option_</i> 7 written.	value parameter can not be accessed or
	EINVAL		d option is invalid at the specified socket socket has been shut down.
	EISCONN		s already connected, and a specified option et while the socket is connected.
	ENOPROTOOPT	The option i	is not supported by the protocol.
	ENOTSOCK	The socket as	rgument does not refer to a socket.
	The setsockopt() function	n may fail if:	
	ENOMEM	There was insufficient memory available for the operation to complete.	
	ENOBUFS	Insufficient i complete the	resources are available in the system to e call.
	ENOSR		insufficient STREAMS resources available ation to complete.
ATTRIBUTES	See attributes(5) for descr	riptions of the	following attributes:
		-	
	ATTRIBUTE TYPE		ATTRIBUTE VALUE
	MT-Level		MT-Safe
SEE ALSO		t(3XNET), ge	MT-Safe tsockopt(3XNET), socket(3XNET),
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	
SEE ALSO	bind(3XNET), endprotoent	t(3XNET), ge	

shutdown – shut down part of a full-duplex connection		
cc [ flag ] filelsocket -lnsl [ library ]		
<pre>int shutdown(int s, int how);</pre>		
The shutdown() call shuts down all or part of a full-duplex connection on the socket associated with <i>s</i> . If <i>how</i> is 0, then further receives will be disallowed. If <i>how</i> is 1, then further sends will be disallowed. If <i>how</i> is 2, then further sends and receives will be disallowed.		
A 0 is returned if the call succeeds, $-1$ if it fails.		
The call succeeds unless:		
EBADF	<i>s</i> is not a valid file descriptor.	
ENOMEM	There was insufficient user memory available for the operation to complete.	
ENOSR	There were insufficient STREAMS resources available for the operation to complete.	
ENOTCONN	The specified socket is not connected.	
ENOTSOCK	<i>s</i> is not a socket.	
See attributes(5) for descriptions of the following attributes:		
	<pre>cc [ flag ] filelsoc int shutdown (int s, int The shutdown () call shuts d associated with s. If how is 0, 4 further sends will be disallow disallowed. A 0 is returned if the call succ The call succeeds unless: EBADF ENOMEM ENOSR ENOTCONN ENOTSOCK</pre>	

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SEE ALSO** connect(3SOCKET), socket(3SOCKET), attributes(5), socket(3HEAD)

**NOTES** The *how* values should be defined constants.

shutdown(3XNET)			
NAME	shutdown – shut down socket send and receive operations		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	<pre>int shutdown(int socket, int how);</pre>		
DESCRIPTION	socket	<i>Sket</i> Specifies the file descriptor of the socket.	
	how	<i>ow</i> Specifies the type of shutdown. The values are as follows:	
		SHUT_RD	Disables further receive operations.
		SHUT_WR	Disables further send operations.
		SHUT_RDWR	Disables further send and receive operations.
		function disables so on the value of the	ubsequent send and/or receive operations on a <i>how</i> argument.
RETURN VALUES	Upon successful completion, shutdown() returns 0. Otherwise, -1 is returned and errno is set to indicate the error.		
ERRORS	The shutdown() function will fail if:		
	EBADF	The so	cket argument is not a valid file descriptor.
	EINVAL	The ho	w argument is invalid.
	ENOTCONN	The so	cket is not connected.
	ENOTSOCK	The so	cket argument does not refer to a socket.
	The shutdown() function may fail if:		
	ENOBUFS		cient resources were available in the system to m the operation.
	ENOSR		were insufficient STREAMS resources available operation to complete.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTR	IBUTE TYPE	ATTRIBUTE VALUE
	MT-Level		MT-Safe
SEE ALSO		d(3XNET), sendto(	, recvfrom(3XNET), recvmsg(3XNET), 3XNET), setsockopt(3XNET),

NAME	slp_api – Service Location Protocol Application Programming Interface
SYNOPSIS	cc [ flag ] filelslp [ library ] #include <slp.h></slp.h>
DESCRIPTION	The slp_api is a C language binding that maps directly into the Service Location Protocol ("SLP") defined by <i>RFC 2614</i> . This implementation requires minimal overhead. With the exception of the SLPDereg() and SLPDelAttrs() functions, which map into different uses of the SLP deregister request, there is one C language function per protocol request. Parameters are for the most part character buffers. Memory management is kept simple because the client allocates most memory and client callback functions are required to copy incoming parameters into memory allocated by the client code. Any memory returned directly from the API functions is deallocated using the SLPFree() function.
	To conform with standard C practice, all character strings passed to and returned through the API are null-terminated, even though the SLP protocol does not use null-terminated strings. Strings passed as parameters are UTF-8 but they may still be passed as a C string (a null-terminated sequence of bytes.) Escaped characters must be encoded by the API client as UTF-8. In the common case of US-ASCII, the usual one byte per character C strings work. API functions assist in escaping and unescaping strings.
	Unless otherwise noted, parameters to API functions and callbacks are non-NULL. Some parameters may have other restrictions. If any parameter fails to satisfy the restrictions on its value, the operation returns a PARAMETER_BAD error.
Syntax for String Parameters	Query strings, attribute registration lists, attribute deregistration lists, scope lists, and attribute selection lists follow the syntax described in <i>RFC 2608</i> . The API reflects the strings passed from clients directly into protocol requests, and reflects out strings returned from protocol replies directly to clients. As a consequence, clients are responsible for formatting request strings, including escaping and converting opaque values to escaped byte-encoded strings. Similarly, on output, clients are required to unescape strings and convert escaped string-encoded opaques to binary. The SLPEscape() and SLPUnescape() functions can be used for escaping SLP reserved characters, but they perform no opaque processing.
	Opaque values consist of a character buffer that contains a UTF-8-encoded string, the first characters of which are the non UTF-8 encoding "\ff". Subsequent characters are the escaped values for the original bytes in the opaque. The escape convention is relatively simple. An escape consists of a backslash followed by the two hexadecimal digits encoding the byte. An example is "\2c" for the byte 0x2c. Clients handle opaque processing themselves, since the algorithm is relatively simple and uniform.
System Properties	The system properties established in slp.conf(4), the configuration file, are accessible through the SLPGetProperty() and SLPSetProperty() functions. The SLPSetProperty() function modifies properties only in the running process, not in the configuration file. Errors are checked when the property is used and, as with parsing the configuration file, are logged at the LOG_INFO priority. Program execution

slp_api(3SLP)			
	continues without interruption by substituting the default for the erroneous parameter. In general, individual agents should rarely be required to override these properties, since they reflect properties of the SLP network that are not of concern to individual agents. If changes are required, system administrators should modify the configuration file.		
	Properties are global to the process, affecting all threads and all handles created with SLPOpen().		
Memory Management	The only API functions that return memory specifically requiring deallocation on the part of the client are SLPParseSrvURL(), SLPFindScope(), SLPEscape(), and SLPUnescape(). Free this memory with SLPFree() when it is no longer needed. Do not free character strings returned by means of the SLPGetProperty() function.		
	Any memory passed to callbacks belongs to the library, and it must not be retained by the client code. Otherwise, crashes are possible. Clients must copy data out of the callback parameters. No other use of the memory in callback parameters is allowed.		
Asynchronous and Incremental Return Semantics	If a handle parameter to an API function is opened asynchronously, the API function calls on the handle to check the other parameters, opens the appropriate operation, and returns immediately. If an error occurs in the process of starting the operation, the error code is returned. If the handle parameter is opened synchronously, the function call is blocked until all results are available, and it returns only after the results are reported through the callback function. The return code indicates whether any errors occurred during the operation.		
	The callback function is called whenever the API library has results to report. The callback code is required to check the error code parameter before looking at the other parameters. If the error code is not SLP_OK, the other parameters may be NULL or otherwise invalid. The API library can terminate any outstanding operation on which an error occurs. The callback code can similarly indicate that the operation should be terminated by passing back SLP_FALSE to indicate that it is not interested in receiving more results. Callback functions are not permitted to recursively call into the API on the same SLPHandle. If an attempt is made to call into the API , the API function returns SLP_HANDLE_IN_USE. Prohibiting recursive callbacks on the same handle simplifies implementation of thread safe code, since locks held on the handle will not be in place during a second outcall on the handle.		
	The total number of results received can be controlled by setting the net.slp.maxResults parameter.		
	On the last call to a callback, whether asynchronous or synchronous, the status code passed to the callback has value SLP_LAST_CALL. There are four reasons why the call can terminate:		
	DA reply received A reply from a DA has been received and therefore nothing more is expected.		

		sip_api(35LP)	
	Multicast terminated	The multicast convergence time has elapsed and the API library multicast code is giving up.	
	Multicast null results	Nothing new has been received during multicast for awhile and the API library multicast code is giving up on that (as an optimization).	
	Maximum results	The user has set the net.slp.maxResults property and that number of replies has been collected and returned.	
Configuration Files	The API library reads slp.conf(4), the operating parameters. You can specify th SLP_CONF_FILE environment variable. refers to is invalid, the API will use the d /etc/inet/slp.conf instead.	e location of this file with the If you do not set this variable, or the file it	
Data Structures	The data structures used by the SLP API are as follows:		
	The URL Lifetime Type		
	<pre>typedef enum {     SLP_LIFETIME_DEFAULT = 10800,     SLP_LIFETIME_MAXIMUM = 65535 } SLPURLLifetime;</pre>		
	The enumeration SLPURLLifetime contains URL lifetime values, in seconds, that are frequently used. SLP_LIFETIME_DEFAULT is 3 hours, while SLP_LIFETIME_MAXIMUM is 18 hours, which corresponds to the maximum size of the lifetime field in SLP messages. Note that on registration SLP_LIFETIME_MAXIMUM causes the advertisement to be continually reregistered until the process exits.		
	The SLPBoolean Type		
	<pre>typedef enum {     SLP_FALSE = 0,     SLP_TRUE = 1 } SLPBoolean;</pre>		
	The enumeration SLPBoolean is used a	s a Boolean flag.	
	The Service URL Structure		
	<pre>typedef struct srvurl {     char *s_pcSrvType;     char *s_pcHost;     int s_iPort;     char *s_pcNetFamily;     char *s_pcSrvPart;</pre>		

Networking Library Functions 539

### slp\_api(3SLP)

#### } SLPSrvURL;

The SLPSrvURL structure is filled in by the SLPParseSrvURL() function with information parsed from a character buffer containing a service URL. The fields correspond to different parts of the URL, as follows: A pointer to a character string containing the service type name, s pcSrvType including naming authority. s pcHost A pointer to a character string containing the host identification information. s iPort The port number, or zero, if none. The port is only available if the transport is IP. s pcNetFamily A pointer to a character string containing the network address family identifier. Possible values are "ipx" for the IPX family, "at" for the Appletalk family, and "", the empty string, for the IP address family. The remainder of the URL, after the host identification. s pcSrvPart The host and port should be sufficient to open a socket to the machine hosting the service; the remainder of the URL should allow further differentiation of the service. The SLPHandle typedef void\* SLPHandle; The SLPHandle type is returned by SLPOpen() and is a parameter to all SLP functions. It serves as a handle for all resources allocated on behalf of the process by the SLP library. The type is opaque. Callbacks Include a function pointer to a callback function specific to a particular API operation in the parameter list when the API function is invoked. The callback function is called with the results of the operation in both the synchronous and asynchronous cases. When the callback function is invoked, the memory included in the callback parameters is owned by the API library, and the client code in the callback must copy out the contents if it wants to maintain the information longer than the duration of the current callback call. Each callback parameter list contains parameters for reporting the results of the operation, as well as an error code parameter and a cookie parameter. The error code parameter reports the error status of the ongoing (for asynchronous) or completed (for synchronous) operation. The cookie parameter allows the client code that starts the operation by invoking the API function to pass information down to the callback without using global variables. The callback returns an SLPBoolean to indicate whether the API library should continue processing the operation. If the value

returned from the callback is SLP\_TRUE, asynchronous operations are terminated. Synchronous operations ignore the return since the operation is already complete.

```
SLPRegReport()
typedef void SLPRegReport(SLPHandle hSLP,
    SLPError errCode,
    void *pvCookie);
```

SLPRegReport() is the callback function to the SLPReg(), SLPDereg(), and SLPDelAttrs() functions. The SLPRegReport() callback has the following parameters:

hSLP	TheSLPHandle() used to initiate the operation.
errCode	An error code indicating if an error occurred during the operation.
pvCookie	Memory passed down from the client code that called the original API function, starting the operation. It may be NULL.

SLPSrvTypeCallback()

```
typedef SLPBoolean SLPSrvTypeCallback(SLPHandle hSLP,
    const char* pcSrvTypes,
    SLPError errCode,
    void *pvCookie);
```

The SLPSrvTypeCallback() type is the type of the callback function parameter to the SLPFindSrvTypes() function. The results are collated when the *hSLP* handle is opened either synchronously or asynchronously. The SLPSrvTypeCallback() callback has the following parameters:

hSLP	The SLPHandle used to initiate the operation.
pcSrvTypes	A character buffer containing a comma-separated, null-terminated list of service types.
errCode	An error code indicating if an error occurred during the operation. The callback should check this error code before processing the parameters. If the error code is other than SLP_OK, then the API library may choose to terminate the outstanding operation.
pvCookie	Memory passed down from the client code that called the original API function, starting the operation. It can be NULL.

#### SLPSrvURLCallback

```
typedef SLPBoolean SLPSrvURLCallback(SLPHandle hSLP,
    const char* pcSrvURL,
    unsigned short usLifetime,
    SLPError errCode,
    void *pvCookie);
```

#### slp\_api(3SLP)

The SLPSrvURLCallback() type is the type of the callback function parameter to the SLPFindSrvs() function. The results are collated, regardless of whether the *hSLP* was opened collated or uncollated. The SLPSrvURLCallback() callback has the following parameters:

- *hSLP* The SLPHandle used to initiate the operation.
- *pcSrvURL* A character buffer containing the returned service URL.
- *usLifetime* An unsigned short giving the life time of the service advertisement. The value must be an unsigned integer less than or equal to SLP LIFETIME MAXIMUM.
- *errCode* An error code indicating if an error occurred during the operation. The callback should check this error code before processing the parameters. If the error code is other than SLP\_OK, then the API library may choose to terminate the outstanding operation.
- *pvCookie* Memory passed down from the client code that called the original API function, starting the operation. It can be NULL.

#### SLPAttrCallback

```
typedef SLPBoolean SLPAttrCallback(SLPHandle hSLP,
    const char* pcAttrList,
    SLPError errCode,
    void *pvCookie);
```

The SLPAttrCallback() type is the type of the callback function parameter to the SLPFindAttrs() function.

The behavior of the callback differs depending upon whether the attribute request was by URL or by service type. If the SLPFindAttrs() operation was originally called with a URL, the callback is called once, in addition to the last call, regardless of whether the handle was opened asynchronously or synchronously. The *pcAttrList* parameter contains the requested attributes as a comma-separated list. It is empty if no attributes match the original tag list.

If the SLPFindAttrs() operation was originally called with a service type, the value of *pcAttrList* and the calling behavior depend upon whether the handle was opened asynchronously or synchronously. If the handle was opened asynchronously, the callback is called every time the API library has results from a remote agent. The *pcAttrList* parameter is collated between calls, and contains a comma-separated list of the results from the agent that immediately returned. If the handle was opened synchronously, the results are collated from all returning agents, the callback is called once, and the *pcAttrList* parameter is set to the collated result.

SLPAttrCallback() callback has the following parameters:

*hSLP* The SLPHandle used to initiate the operation.

	pcAttrList		ng a comma-separated and null-terminated signments, in SLP wire format.	
	errCode	An error code indicating if an error occurred during the operation. The callback should check this error code before processing the parameters. If the error code is other than SLP_OK, then the API library may choose to terminate the outstanding operation.		
	pvCookie	Memory passed down from function, starting the opera	n the client code that called the original API tion. It can be NULL.	
ERRORS	An interface	that is part of the SLP API m	ay return one of the following values.	
	SLP_LAST_	CALL	The SLP_LAST_CALL code is passed to callback functions when the API library has no more data for them and therefore no further calls will be made to the callback on the currently outstanding operation. The callback uses this to signal the main body of the client code that no more data will be forthcoming on the operation, so that the main body of the client code can break out of data collection loops. On the last call of a callback during both a synchronous and asynchronous call, the error code parameter has value SLP_LAST_CALL, and the other parameters are all NULL. If no results are returned by an API operation, then only one call is made, with the error parameter set to SLP_LAST_CALL.	
	SLP_OK		The SLP_OK code indicates that the no error occurred during the operation.	
	SLP_LANGU	AGE_NOT_SUPPORTED	No DA or SA has service advertisement information in the language requested, but at least one DA or SA might have information for that service in another language.	
	SLP_PARSE	_ERROR	The SLP message was rejected by a remote SLP agent. The API returns this error only when no information was retrieved, and at least one SA or DA indicated a protocol error. The data supplied through the API may be malformed or damaged in transit.	
	SLP_INVAL	ID_REGISTRATION	The API may return this error if an attempt to register a service was rejected by all DAs because of a malformed URL or attributes.	

# slp\_api(3SLP)

	SLP does not return the error if at least one DA accepts the registration.
SLP_SCOPE_NOT_SUPPORTED	The API returns this error if the UA or SA has been configured with the net.slp.useScopes list of scopes and the SA request did not specify one or more of these allowable scopes, and no others. It may also be returned by a DA if the scope included in a request is not supported by a DA.
SLP_AUTHENTICATION_ABSENT	This error arises when the UA or SA failed to send an authenticator for requests or registrations when security is enabled and thus required.
SLP_AUTHENTICATION_FAILED	This error arises when a authentication on an SLP message received from a remote SLP agent failed.
SLP_INVALID_UPDATE	An update for a nonexisting registration was issued, or the update includes a service type or scope different than that in the initial registration.
SLP_REFRESH_REJECTED	The SA attempted to refresh a registration more frequently than the minimum refresh interval. The SA should call the appropriate API function to obtain the minimum refresh interval to use.
SLP_NOT_IMPLEMENTED	An outgoing request overflowed the maximum network MTU size. The request should be reduced in size or broken into pieces and tried again.
SLP_BUFFER_OVERFLOW	An outgoing request overflowed the maximum network MTU size. The request should be reduced in size or broken into pieces and tried again.
SLP_NETWORK_TIMED_OUT	When no reply can be obtained in the time specified by the configured timeout interval, this error is returned.
SLP_NETWORK_INIT_FAILED	If the network cannot initialize properly, this error is returned.
SLP_MEMORY_ALLOC_FAILED	If the API fails to allocate memory, the operation is aborted and returns this.

$sip_api(solit)$	slp	_api(3SLF	')
------------------	-----	-----------	----

		sip_api(SSLP)
	SLP_PARAMETER_BAD	If a parameter passed into an interface is bad, this error is returned.
	SLP_NETWORK_ERROR	The failure of networking during normal operations causes this error to be returned.
	SLP_INTERNAL_SYSTEM_ERROR	A basic failure of the API causes this error to be returned. This occurs when a system call or library fails. The operation could not recover.
	SLP_HANDLE_IN_USE	In the C API, callback functions are not permitted to recursively call into the API on the same SLPHandle, either directly or indirectly. If an attempt is made to do so, this error is returned from the called API function
LIST OF	SLPOpen()	open an SLP handle
ROUTINES	SLPClose()	close an open SLP handle
	SLPReg()	register a service advertisement
	SLPDereg()	deregister a service advertisement
	SLPDelAttrs()	delete attributes
	SLPFindSrvTypes()	return service types
	SLPFindSrvs()	return service URLs
	SLPFindAttrs()	return service attributes
	SLPGetRefreshInterval()	return the maximum allowed refresh interval for SAs
	SLPFindScopes()	return list of configured and discovered scopes
	SLPParseSrvURL()	parse service URL
	SLPEscape()	escape special characters
	SLPUnescape()	translate escaped characters into UTF-8
	SLPGetProperty()	return SLP configuration property
	SLPSetProperty()	set an SLP configuration property
	<pre>slp_strerror()</pre>	map SLP error code to message
	SLPFree()	free memory
ENVIRONMENT	When SLP_CONF_FILE is set, use this fil	le for configuration.
VARIABLES ATTRIBUTES	See attributes(5) for descriptions of the	ne following attributes:

Networking Library Functions 545

#### slp\_api(3SLP)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWslpu
CSI	CSI-enabled
Interface Stability	Standard
MT-Level	Safe

#### **SEE ALSO** slpd(1M), slp.conf(4), slpd.reg(4), attributes(5)

System Administration Guide: Resource Management and Network Services

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

NAME	SLPClose – close an open SLP handle			
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>			
	<pre>void SLPClose(SLPHandle phSLP);</pre>			
DESCRIPTION	The SLPClose() function frees all resources associated with the handle. If the handle is invalid, the function returns silently. Any outstanding synchronous or asynchronous operations are cancelled, so that their callback functions will not be called any further			
PARAMETERS	phSLP An SLPHandle handle	returned from a call to SPLOpen().		
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).			
EXAMPLES	<b>EXAMPLE 1</b> Using SLPClose()			
	The following example will free all resource	es associated the handle:		
	SLPHandle hslp SLPCLose(hslp);			
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file for configuration.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ATTRIBUTE TYPE			
	ATTRIBUTE TYPE Availability	ATTRIBUTE VALUE SUNWslpu		
SEE ALSO		SUNWslpu		
SEE ALSO	Availability	SUNWslpu slpd.reg(4),attributes(5)		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4),	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), <i>System Administration Guide: Resource Manag</i> Kempf, J. and Guttman, E., <i>RFC 2614, An A</i>	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		

SLPDelAttrs(3SLP)				
NAME	SLPDelAttrs – delete attributes			
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>			
	<pre>SLPError SLPDelAttrs(SLPHandle hSLP, const char *pcURL, const char *pcAttrs, SLPRegReport *callback, void *pvCookie);</pre>			
DESCRIPTION	The SLPDelAttrs() function deletes the selected attributes in the locale of the SLPHandle. If no error occurs, the return value is 0. Otherwise, one of the SLPError codes is returned.			
PARAMETERS	hSLP	The language specific S cannot be NULL.	LPHandle to use to delete attributes. It	
	pcURL	The URL of the advertis deleted. It cannot be NU	sement from which the attributes should be LL.	
	<i>pcAttrs</i> A comma-separated list of attribute ids for the attributes to deregister.			
	callback	A callback to report the NULL.	operation's completion status. It cannot be	
	pvCookie	Memory passed to the c	callback code from the client. It cannot be	
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).			
EXAMPLES	EXAMPLE 1 Deleting	Attributes		
	0	example to delete the lo er:lpr://serv/queve	cation and dpi attributes for the URL 1	
	SLPHandle hSLP; SLPError err; SLPRegReport report;			
	err = SLPDelAttrs(hSLP, "service:printer:lpr://serv/queue1", "location,dpi", report, NULL);			
ENVIRONMENT VARIABLES	SLP_CONF_FILE	When set, use this file	e for configuration.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ΑΤΤ	RIBUTE TYPE	ATTRIBUTE VALUE	
	Availability		SUNWslpu	
SEE ALSO			slpd.reg(4), attributes(5) gement and Network Services	

SLPDelAttrs(3SLP)

SLPDereg(3SLP)				
NAME	SLPDereg – deregister the SLP advertisement			
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>			
	<pre>SLPError SLPDereg(SLPHandle hSLP, const char *pcURL, SLPRegReport</pre>			
DESCRIPTION	The SLPDereg() function deregisters the advertisement for URL <i>pcURL</i> in all scopes where the service is registered and in all language locales, not just the locale of the SLPHandle. If no error occurs, the return value is 0. Otherwise, one of the SLPError codes is returned.			
PARAMETERS	<i>hSLP</i> The language specific SLPHandle to use for deregistering. <i>hSLP</i> cannot be NULL.			
	pcURL	The URL to deregister.	The value of <i>pcURL</i> cannot be NULL.	
	<i>callback</i> A callback to report the operation completion status. <i>callback</i> cannot be NULL.			
	pvCookie	Memory passed to the obe NULL.	callback code from the client. <i>pvCookie</i> can	
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).			
EXAMPLES	EXAMPLE 1 Using SL	PDereg()		
		Use the following example to deregister the advertisement for the URL "service:ftp://csserver":		
	SLPerror err; SLPHandle hSLP; SLPRegReport regreport;			
	err = SLPDereg(hSLP, "service:ftp://csserver", regreport, NULL);			
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file for configuration.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ATT		ATTRIBUTE VALUE	
	Availability		SUNWslpu	
SEE ALSO	<pre>slpd(1M), slp_a;</pre>	pi(3SLP), slp.conf(4),	<pre>slpd.reg(4), attributes(5)</pre>	
	System Administrat	ion Guide: Resource Manag	gement and Network Services	
		ns, C., Veizades, J., and I met Society, June 1999.	Day, M., RFC 2608, Service Location Protocol,	

SLPDereg(3SLP)

## SLPEscape(3SLP)

NAME	SLPEscape – escapes SLP reserved characters			
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>			
	<pre>SLPError SLPEscape(const char *pcInBuf, char** ppcOutBuf, SLPBoolean</pre>			
DESCRIPTION	The SLPEscape() function processes the input string in <i>pclnbuf</i> and escapes any SLP reserved characters. If the <i>isTag</i> parameter is SLPTrue, it then looks for bad tag characters and signals an error if any are found by returning the SLP_PARSE_ERROR code. The results are put into a buffer allocated by the API library and returned in the <i>ppcOutBuf</i> parameter. This buffer should be deallocated using SLPFree(3SLP) when the memory is no longer needed.			
PARAMETERS	pcInBuf	Pointer to the input buf	fer to process for escape characters.	
	<i>ppcOutBuf</i> Pointer to a pointer for the output buffer with the SLP reserved characters escaped. It must be freed using SLPFree() when the memory is no longer needed.			
	isTag	When true, checks the in	nput buffer for bad tag characters.	
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).			
EXAMPLES	EXAMPLE 1 Converting Attribute Tags			
	The following exar the wire format:	nple shows how to conve	ert the attribute tag ,tag-example, to on	
	SLPError err; char* escaped Chars;			
	<pre>err = SLPEscape(",tag-example,", &amp;escapedChars, SLP_TRUE);</pre>			
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file for configuration.			
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:			
	ATTF	RIBUTE TYPE	ATTRIBUTE VALUE	
	Availability		SUNWslpu	
SEE ALSO	<pre>slpd(1M), slp_ap attributes(5)</pre>	pi(3SLP), SLPFree(3SLF	<pre>P), slp.conf(4), slpd.reg(4),</pre>	
	System Administrat	ion Guide: Resource Manag	gement and Network Services	
		Guttman, E., Perkins, C., Veizades, J., and Day, M., <i>RFC 2608, Service Location Protocol, Version 2</i> , The Internet Society, June 1999.		

SLPEscape(3SLP)

SLPFindAttrs(	3SLP)	
---------------	-------	--

NAME	SLPFindAttrs – return service attributes	
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>	
	<pre>SLPError SLPFindAttrs(SLPHandle hSLP, const char *pcURL, const char *pcScopeList, const char *pcAttrIds, SLPAttrCallback *callback, void *pvCookie);</pre>	
DESCRIPTION	The SLPFindAttrs() function returns service attributes matching the attribute tags for the indicated full or partial URL. If <i>pcURL</i> is a complete URL, the attribute information returned is for that particular service in the language locale of the SLPHandle. If <i>pcURL</i> is a service type, then all attributes for the service type are returned, regardless of the language of registration. Results are returned through the <i>callback</i> parameter.	
		ed with an SLP attribute request filter string parameter, the syntax of I in <i>RFC 2608</i> . If the filter string is the empty string, "", all attributes
	If an error occurs i	n starting the operation, one of the SLPError codes is returned.
PARAMETERS	hSLP	The language-specific SLPHandle on which to search for attributes. It cannot be NULL.
	pcURL	The full or partial URL. See <i>RFC 2608</i> for partial URL syntax. It cannot be NULL.
	pcScopeList	A pointer to a char containing a comma-separated list of scope names. It cannot be NULL or an empty string, " ".
	<i>pcAttrIds</i> The filter string indicating which attribute values to return. Use empty string " " to indicate all values. Wildcards matching all attribute ids having a particular prefix or suffix are also possible. It cannot be NULL.	
	callback	A callback function through which the results of the operation are reported. It cannot be NULL.
	<i>pvCookie</i> Memory passed to the callback code from the client. It may be NULL.	
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).	
EXAMPLES	EXAMPLE 1 Returnin	g Service Attributes for a Specific URL
		example to return the attributes "location" and "dpi" for the printer:lpr://serv/queue1" through the callback
	SLPHandle hSLP; SLPAttrCallback a SLPError err;	ttrReturn;

554 man pages section 3: Networking Library Functions • Last Revised 17 Nov 1999

### SLPFindAttrs(3SLP)

	<b>EXAMPLE 1</b> Returning Service Attributes for a Specific URL ( <i>Continued</i> )		
<pre>err = SLPFindAttrs(hSLP "service:printer:lpr://serv/queuel", "default", "location,dpi", attrReturn, err);</pre>			
<b>EXAMPLE 2</b> Returning Service Attributes for All URLs of a Specific Type	<b>EXAMPLE 2</b> Returning Service Attributes for All URLs of a Specific Type		
Use the following example to return the attributes "location" and "dpi" for service URLs having type "service:printer:lpr":	Use the following example to return the attributes "location" and "dpi" for all service URLs having type "service:printer:lpr":		
<pre>err = SLPFindAttrs(hSLP, "service:printer:lpr", "default", "location, pi", attrReturn, NULL);</pre>			
<b>ENVIRONMENT</b> SLP_CONF_FILE When set, use this file for configuration.			
ATTRIBUTES See attributes(5) for descriptions of the following attributes:			
ATTRIBUTE TYPE ATTRIBUTE VALUE			
ATTRIBUTE TYPE     ATTRIBUTE VALUE       Availability     SUNWslpu			
SEE ALSO slpd(1M), slp_api(3SLP), slp.conf(4), slpd.reg(4), attributes(5)			
System Administration Guide: Resource Management and Network Services			
Kempf, J. and Guttman, E., <i>RFC 2614, An API for Service Location</i> , The Internet S June 1999.	Society,		

## SLPFindScopes(3SLP)

NAME	SLPFindScopes – return list of configured and discovered scopes		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPFindScopes(SLPHandle hSLP, char** ppcScopes);</pre>		
DESCRIPTION	The SLPFindScopes() function sets the <i>ppcScopes</i> parameter to a pointer to a comma-separated list including all available scope names. The list of scopes comes from a variety of sources: the configuration file, the net.slp.useScopes property and the net.slp.DAAddresses property, DHCP, or through the DA discovery process. If there is any order to the scopes, preferred scopes are listed before less desirable scopes. There is always at least one string in the array, the default scope, DEFAULT.		
	If no error occurs, appropriate error o		Irns SLP_OK, otherwise, it returns the
PARAMETERS	hSLP	The SLPHandle on wh NULL.	ich to search for scopes. <i>hSLP</i> cannot be
	ppcScopes	upon return. The buffer	inter into which the buffer pointer is placed r is null-terminated. The memory should be ee(). See SLPFree(3SLP)
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	EXAMPLE 1 Finding Configured or Discovered Scopes		
	Use the following	example to find configur	ed or discovered scopes:
	SLPHandle hSLP; char *ppcScopes; SLPError err;		
	error = SLPFindScopes(hSLP, & ppcScopes);		
ENVIRONMENT VARIABLES	SLP_CONF_FILE	When set, use this file	e for configuration.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTF	RIBUTE TYPE	ATTRIBUTE VALUE
	Availability		SUNWslpu
SEE ALSO	<pre>slpd(1M), slp_api(3SLP), SLPFree(3SLP), slp.conf(4), slpd.reg(4), attributes(5) System Administration Guide: Resource Management and Network Services</pre>		
		ion Guide: Resource Mana	gement and Network Services

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

SLPFindSrvs(3SLP)			
NAME	SLPFindSrvs – return service URLs		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPFindSrvs(SLPHandle hSLP, const char *pcServiceType, const char *pcScopeList, const char *pcSearchFilter, SLPSrvURLCallback *callback, void *pvCookie);</pre>		
DESCRIPTION	The SLPFindSrvs() function issues a request for SLP services. The query is for services on a language-specific SLPHandle. It returns the results through the <i>callback</i> . The parameters will determine the results.		
	If an error occurs in starting t	he operation, one of the SLPError codes is returned.	
PARAMETERS	hSLP	The language-specific SLPHandle on which to search for services. It cannot be NULL.	
	pcServiceType	The service type string for the request. The <i>pcServiceType</i> can be discovered by a call to SLPSrvTypes(). Examples of service type strings include	
		"service:printer:lpr"	
		or	
		"service:nfs"	
		<i>pcServiceType</i> cannot be NULL.	
	pcScopeList	A pointer to a char containing a comma-separated list of scope names. It cannot be NULL or an empty string, "".	
	pcSearchFilter	A query formulated of attribute pattern matching expressions in the form of a LDAPv3 search filter. See <i>RFC</i> 2254. If this filter is empty, "", all services of the requested type in the specified scopes are returned. It cannot be NULL.	
	callback	A callback through which the results of the operation are reported. It cannot be NULL.	
	<i>pvCookie</i> Memory passed to the callback code from the client. It can be NULL.		
ERRORS	This function or its callback m slp_api(3SLP).	nay return any SLP error code. See the ERRORS section in	
EXAMPLES	EXAMPLE 1 Using SLPFindSrvs	5()	
	The following example finds a protocol with the dpi attribute	all advertisements for printers supporting the LPR e 300 in the default scope:	

	<b>EXAMPLE 1</b> Using SLPFindSrvs() (Contin	nued)
	SLPError err; SLPHandle hSLP; SLPSrvURLCallback srvngst;	
	<pre>err = SLPFindSrvs(hSLP, "service:printer:lpr", "default", "(dpi=300)", srvngst, NULL);</pre>	
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file	-
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	
	Availability	SUNWslpu
SEE ALSO	<pre>slpd(1M), slp_api(3SLP), slp.conf(4),</pre>	<pre>slpd.reg(4), attributes(5)</pre>
	System Administration Guide: Resource Manag	gement and Network Services
	Howes, T. RFC 2254, The String Representation Society. 1997.	on of LDAP Search Filters. The Internet
	Guttman, E., Perkins, C., Veizades, J., and E <i>Version 2</i> . The Internet Society. June 1999.	Day, M. RFC 2608, Service Location Protocol,
	Kempf, J. and Guttman, E. <i>RFC 2614, An Al</i> June 1999.	PI for Service Location. The Internet Society.

# SLPFindSrvTypes(3SLP)

NAME	SLPFindSrvTypes – find service types		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPFindSrvTypes(SLPHandle hSLP, const char  *pcNamingAuthority, const char *pcScopeList, SLPSrvTypeCallback  *callback, void *pvCookie);</pre>		
DESCRIPTION	The SLPFindSrvTypes() function issues an SLP service type request for service types in the scopes indicated by the pcScopeList. The results are returned through the <i>callback</i> parameter. The service types are independent of language locale, but only for services registered in one of the scopes and for the indicated naming authority.		
	If the naming authority is "*", then results are returned for all naming authorities. If the naming authority is the empty string, "", then the default naming authority, IANA, is used. IANA is not a valid naming authority name; the SLP_PARAMETER_BAD error code will be returned if you include it explicitly.		
	The service type names are re following format:	turned with the naming authority included in the	
	service-type "." naming-aut which case, just the service ty	chorityunless the naming authority is the default, in pe name is returned.	
	If an error occurs in starting t	he operation, one of the SLPError codes is returned.	
PARAMETERS	hSLP	The SLPHandle on which to search for types. It cannot be NULL.	
	pcNamingAuthority	The naming authority to search. Use "*" to search all naming authorties; use the empty string "" to search the default naming authority. It cannot be NULL.	
	pcScopeList	A pointer to a char containing a comma-separated list of scope names to search for service types. It cannot be NULL or an empty string, "".	
	callback	A callback through which the results of the operation are reported. It cannot be NULL.	
	pvCookie	Memory passed to the callback code from the client. It can be NULL.	
ERRORS	This function or its callback m slp_api(3SLP).	nay return any SLP error code. See the ERRORS section in	
EXAMPLES	<b>EXAMPLE 1</b> Using SLPFindSrv7	Types()	
	The following example finds a naming authority:	all service type names in the default scope and default	

#### SLPFindSrvTypes(3SLP)

```
EXAMPLE 1 Using SLPFindSrvTypes() (Continued)
SLPError err;
SLPHandle hSLP;
SLPSrvTypeCallback findsrvtypes;
err = SLPFindSrvTypes(hSLP, "", "default", findsrvtypes, NULL);
```

SLP\_CONF\_FILE When set, use this file for configuration.

ENVIRONMENT VARIABLES ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWslpu

**SEE ALSO** | slpd(1M), slp\_api(3SLP), slp.conf(4), slpd.reg(4), attributes(5)

System Administration Guide: Resource Management and Network Services

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

Howes, T., *RFC* 2254, *The String Representation of LDAP Search Filters*, The Internet Society, 1997.

## SLPFree(3SLP)

NAME	SLPFree – frees memory		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPFree(void *pvMem);</pre>		
DESCRIPTION	The SLPFree() function frees memory returned from SLPParseSrvURL(), SLPFindScopes(), SLPEscape(), and SLPUnescape().		
PARAMETERS	pvMem       A pointer to the storage allocated by the SLPParseSrvURL(),         SLPFindScopes(), SLPEscape(), and SLPUnescape()         functions. pvMem is ignored if its value is NULL.		
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	EXAMPLE 1 Using SL	PFree()	
		nple illustrates how to ca y allocated memory.	ll SLPFree(). It assumes that $SrvURL$
	SLPerror err;		
	err = SLPFree((vo:	id*) SrvURL);	
ENVIRONMENT VARIABLES	SLP_CONF_FILE	When set, use this file	e for configuration.
ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATTF	RIBUTE TYPE	ATTRIBUTE VALUE
	<b>ATTF</b> Availability	RIBUTE TYPE	ATTRIBUTE VALUE SUNWslpu
SEE ALSO	Availability	cape(3SLP), SLPFindSc	
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI	cape(3SLP), SLPFindSc LP), slp_api(3SLP), slj	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP),
SEE ALSO	Availability slpd(1M), SLPEsc SLPUnescape(3SI <i>System Administrat</i> Guttman, E., Perki	cape(3SLP), SLPFindSc LP ), slp_api(3SLP), slj ion Guide: Resource Manaş	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5)
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI System Administrat Guttman, E., Perki Version 2, The Inter	cape(3SLP), SLPFindSc LP), slp_api(3SLP), sl <i>ion Guide: Resource Manaş</i> ns, C., Veizades, J., and E met Society, June 1999.	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5) gement and Network Services
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI System Administrat Guttman, E., Perki Version 2, The Inter Kempf, J. and Gutt	cape(3SLP), SLPFindSc LP), slp_api(3SLP), sl <i>ion Guide: Resource Manaş</i> ns, C., Veizades, J., and E met Society, June 1999.	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5) gement and Network Services Day, M., RFC 2608, Service Location Protocol,
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI System Administrat Guttman, E., Perki Version 2, The Inter Kempf, J. and Gutt	cape(3SLP), SLPFindSc LP), slp_api(3SLP), sl <i>ion Guide: Resource Manaş</i> ns, C., Veizades, J., and E met Society, June 1999.	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5) gement and Network Services Day, M., RFC 2608, Service Location Protocol,
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI System Administrat Guttman, E., Perki Version 2, The Inter Kempf, J. and Gutt	cape(3SLP), SLPFindSc LP), slp_api(3SLP), sl <i>ion Guide: Resource Manaş</i> ns, C., Veizades, J., and E met Society, June 1999.	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5) gement and Network Services Day, M., RFC 2608, Service Location Protocol,
SEE ALSO	Availability slpd(1M), SLPEso SLPUnescape(3SI System Administrat Guttman, E., Perki Version 2, The Inter Kempf, J. and Gutt	cape(3SLP), SLPFindSc LP), slp_api(3SLP), sl <i>ion Guide: Resource Manaş</i> ns, C., Veizades, J., and E met Society, June 1999.	SUNWslpu opes(3SLP), SLPParseSrvURL(3SLP), p.conf(4), slpd.reg(4), attributes(5) gement and Network Services Day, M., RFC 2608, Service Location Protocol,

## SLPGetProperty(3SLP)

NAME	SLPGetProperty – return SLP configuration property			
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>			
	<pre>const char* SLPGetProperty(const char* pcName);</pre>			
DESCRIPTION	The SLPGetProperty() function returns the value of the corresponding SLP property name, or NULL, if none. If there is no error, SLPGetProperty() returns a pointer to the property value. If the property was not set, it returns the empty string, "". If an error occurs, SLPGetProperty() returns NULL. The returned string should not be freed.			
PARAMETERS	<i>pcName</i> A null-terminated string with the	ne property name. <i>pcName</i> cannot be NULL.		
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).			
EXAMPLES	<b>EXAMPLE 1</b> Using SLPGetProperty()			
	Use the following example to return a list o	f configured scopes:		
	const char* useScopes			
	<pre>useScopes = SLPGetProperty("net.slp.useScopes = SLPGetPropere</pre>	copes");		
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file	e for configuration.		
ATTRIBUTES	See attributes(5) for descriptions of the	See attributes(5) for descriptions of the following attributes:		
		Ŭ		
	ATTRIBUTE TYPE			
	ATTRIBUTE TYPE Availability	ATTRIBUTE VALUE SUNWslpu		
SEE ALSO		SUNWslpu		
SEE ALSO	Availability	SUNWslpu slpd.reg(4),attributes(5)		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4),	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4), System Administration Guide: Resource Manag Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services		

# SLPGetRefreshInterval(3SLP)

NAME	SLPGetRefreshInterval – return the maximum allowed refresh interval		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>int SLPGetRefreshIntervalvoid););</pre>		
DESCRIPTION	The SLPGetRefreshInterval () function returns the maximum across all DAs of the min-refresh-interval attribute. This value satisfies the advertised refresh interval bounds for all DAs. If this value is used by the SA, it assures that no refresh registration will be rejected. If no DA advertises a min-refresh-interval attribute, a value of 0 is returned. If an error occurs, an SLP error code is returned.		
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	<b>EXAMPLE 1</b> Using SLPGetRefreshInterval	()	
	Use the following example to return the ma	eximum valid refresh interval for SA:	
	int minrefresh		
	<pre>minrefresh = SLPGetRefreshInterval( );</pre>		
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file	e for configuration.	
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
		ATTRIBUTE VALUE	
	ATTRIBUTE TYPE Availability	ATTRIBUTE VALUE SUNWslpu	
SEE ALSO		SUNWslpu slpd.reg(4),attributes(5)	
SEE ALSO	Availability slpd(1M), slp_api(3SLP), slp.conf(4),	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	
SEE ALSO	Availability         slpd(1M), slp_api(3SLP), slp.conf(4),         System Administration Guide: Resource Manage         Kempf, J. and Guttman, E., RFC 2614, An A	SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	

NAME	SLPOpen – open an SLP handle		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPOpen(const char *pcLang, SLPBoolean isAsync, SLPHandle  *phSLP);</pre>		
DESCRIPTION	The SLPOpen() function returns a SLPHandle handle in the <i>phSLP</i> parameter for the language locale passed in as the <i>pcLang</i> parameter. The client indicates if operations on the handle are to be synchronous or asynchronous through the <i>isAsync</i> parameter. The handle encapsulates the language locale for SLP requests issued through the handle, and any other resources required by the implementation. SLP properties are not encapsulated by the handle, they are global. The return value of the function is an SLPError code indicating the status of the operation. Upon failure, the <i>phSLP</i> parameter is NULL.		
	An SLPHandle can only be used for one SLP API operation at a time. If the original operation was started asynchronously, any attempt to start an additional operation on the handle while the original operation is pending results in the return of an SLP_HANDLE_IN_USE error from the API function. The SLPClose() function terminates any outstanding calls on the handle.		
PARAMETERS	pcLang	A pointer to an array of characters containing the language tag set forth in <i>RFC 1766</i> for the natural language locale of requests issued on the handle. This parameter cannot be NULL.	
	isAsync	An SLPBoolean indicating whether or not the SLPHandle should be opened for an asynchronous operation.	
	phSLP	A pointer to an SLPHandle in which the open SLPHandle is returned. If an error occurs, the value upon return is NULL.	
ERRORS	This function or its slp_api(3SLP).	s callback may return any SLP error code. See the ERRORS section in	
EXAMPLES	EXAMPLE 1 Using SI	POpen()	
	Use the following locale:	example to open a synchronous handle for the German ("de")	
	SLPHandle HSLP; SLPError err;		
	err = SLPOpen("de	", SLP_FALSE, &hSLP)	
ENVIRONMENT VARIABLES	SLP_CONF_FILE	When set, use this file for configuration.	
ATTRIBUTES	See attributes(	5) for descriptions of the following attributes:	

Networking Library Functions 565

## SLPOpen(3SLP)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWslpu	
SEE ALSO	<pre>slpd(1M), slp_api(3SLP), slp.conf(4), slpd.reg(4), attributes(5)</pre>		
	System Administration Guide: Resource Management and Network Services		
	Alvestrand, H., <i>RFC 1766, Tags for the Identification of Languages,</i> Network Working Group, March 1995.		
	Kempf, J. and Guttman, E., <i>RFC 2614, An A</i> June 1999.	PI for Service Location, The Internet Society,	

### SLPParseSrvURL(3SLP)

NAME	SLPParseSrvURL – parse service URL		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPParseSrvURL(const char *pcSrvURL, SLPSrvURL**     ppSrvURL);</pre>		
DESCRIPTION	The SLPParseSrvURL() routine parses the URL passed in as the argument into a service URL structure and returns it in the <i>ppSrvURL</i> pointer. If a parser error occurs, returns SLP_PARSE_ERROR. The structure returned in <i>ppSrvURL</i> should be freed with SLPFree(). If the URL has no service part, the s_pcSrvPart string is the empty string, "", that is, it is not NULL. If <i>pcSrvURL</i> is not a service: URL, then the s_pcSrvType field in the returned data structure is the URL's scheme, which might not be the same as the service type under which the URL was registered. If the transport is IP, the s_pcNetFamily field is the empty string.		
	If no error occurs, the return value is the SLP_OK. Otherwise, if an error occurs, one of the SLPError codes is returned.		
PARAMETERS	pcSrvURL		buffer containing the null terminated URL tructively modified to produce the output NULL.
	ppSrvURL	A pointer to a ponter for parsed URL. It may not	r the SLPSrvURL structure to receive the be NULL.
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	EXAMPLE 1 Using SLPParseSrvURL()		
	The following example uses the SLPParseSrvURL() function to parse the service URL service:printer:lpr://serv/queue1:		
	SLPSrvURL* surl; SLPError err;		
	err = SLPParseSrvURL("service:printer:lpr://serv/queuel", &surl);		
ENVIRONMENT VARIABLES	SLP_CONF_FILE	F_FILE When set, use this file for configuration.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ΑΤΤ		ATTRIBUTE VALUE
	Availability		SUNWslpu
	[		
SEE ALSO	<pre>slpd(1M), slp_a</pre>	pi(3SLP), slp.conf(4),	<pre>slpd.reg(4), attributes(5)</pre>
	System Administrat	ion Guide: Resource Mana	gement and Network Services

Networking Library Functions 567

SLPParseSrvURL(3SLP)

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

NAME	SLPReg – register an SLP advertisement		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPReg(SLPHandle hSLP, const char *pcSrvURL, const unsigned short usLifetime , const char *pcSrvType, const char *pcAttrs, SLPBoolean fresh, SLPRegReport callback, void *pvCookie);</pre>		
DESCRIPTION	The SLPReg() function registers the URL in <i>pcSrvURL</i> having the lifetime <i>usLifetime</i> with the attribute list in <i>pcAttrs</i> . The <i>pcAttrs</i> list is a comma-separated list of attribute assignments in on-the-wire format (including escaping of reserved characters). The <i>sLifetime</i> parameter must be nonzero and less than or equal to SLP_LIFETIME_MAXIMUM. If the fresh flag is SLP_TRUE, then the registration is new, the SLP protocol <i>fresh</i> flag is set, and the registration replaces any existing registrations.		
	The <i>pcSrvType</i> parameter is a service type name and can be included for service URLs that are not in the service: scheme. If the URL is in the service: scheme, the <i>pcSrvType</i> parameter is ignored. If the fresh flag is SLP_FALSE, then an existing registration is updated. Rules for new and updated registrations, and the format for <i>pcAttrs</i> and <i>pcScopeList</i> , can be found in <i>RFC 2608</i> . Registrations and updates take place in the language locale of the <i>hSLP</i> handle.		
	The API library is required to perform the operation in all scopes obtained through configuration.		
PARAMETERS	hSLP	The language specific SLPHandle on which to register the advertisement. <i>hSLP</i> cannot be NULL.	
	pcSrvURL	The URL to register. The value of <i>pcSrvURL</i> cannot be NULL or the empty string.	
	usLifetime	An unsigned short giving the life time of the service advertisement, in seconds. The value must be an unsigned integer less than or equal to SLP_LIFETIME_MAXIMUM.	
	pcSrvType	The service type. If <i>pURL</i> is a service: URL, then this parameter is ignored. <i>pcSrvType</i> cannot be NULL.	
	pcAttrs	A comma-separated list of attribute assignment expressions for the attributes of the advertisement. <i>pcAttrs</i> cannot be NULL. Use the empty string, "", to indicate no attributes.	
	fresh	An SLPBoolean that is SLP_TRUE if the registration is new or SLP_FALSE if it is a reregistration.	
	callback	A callback to report the operation completion status. <i>callback</i> cannot be NULL.	
	pvCookie	Memory passed to the callback code from the client. <i>pvCookie</i> can be NULL.	

## SLPReg(3SLP)

ERRORS	This function or its callback may return any slp_api(3SLP).	SLP error code. See the ERRORS section in	
EXAMPLES	EXAMPLE 1 An Initial Registration		
	The following example shows an initial registration for the "service:video://bldg15" camera service for three hours:		
	<pre>SLPError err; SLPHandle hSLP; SLPRegReport regreport; err = SLPReg(hSLP, "service:video://bldg1 10800, "", "(location=B15-corridor), (scan-rate=100)", SLP_TRUE, regRpt, NULL);</pre>		
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file for configuration.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	Availability	SUNWslpu	
SEE ALSO	<pre>slpd(1M), slp_api(3SLP), slp.conf(4), slpd.reg(4), attributes(5) System Administration Guide: Resource Management and Network Services</pre>		
	Guttman, E., Perkins, C., Veizades, J., and Day, M., <i>RFC 2608, Service Location Protocol, Version 2</i> , The Internet Society, June 1999.		
	Kempf, J. and Guttman, E., <i>RFC 2614, An API for Service Location</i> , The Internet Society, June 1999.		

## SLPSetProperty(3SLP)

SLPSetProperty – set an SLP configuration property		
<pre>#include <slp.h></slp.h></pre>		
<pre>void SLPSetProperty(const char *pcName, const char *pcValue);</pre>		
The SLPSetProperty() function sets the value of the SLP property to the new value. The <i>pcValue</i> parameter contains the property value as a string.		
cannot		
cannot		
This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLE 1 Setting a Configuration Property		
Society,		

slp\_strerror(3SLP)

NAME	slp_strerror – map SLP error codes to messages		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>const char* slp_strerror(SLPError err_code);</pre>		
DESCRIPTION	The slp_strerror() function maps err_code to a string explanation of the error. The returned string is owned by the library and must not be freed.		
PARAMETERS	<i>err_code</i> An SLP error code.		
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	EXAMPLE 1 Using slp_sterror()		
	The following example returns the message	e that corresponds to the error code:	
	<pre>SLPError error; const char* msg; msg = slp_streerror(err);</pre>		
ENVIRONMENT VARIABLES	SLP_CONF_FILE When set, use this file for configuration.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE		
		ATTRIBUTE VALUE	
	Availability	ATTRIBUTE VALUE SUNWslpu	
SEE ALSO		SUNWslpu slpd.reg(4), attributes(5) gement and Network Services	

NAME	SLPUnescape – translate escaped characters into UTF-8		
SYNOPSIS	<pre>#include <slp.h></slp.h></pre>		
	<pre>SLPError SLPUnescape(const char *pcInBuf, char** ppcOutBuf, SLPBoolean isTag);</pre>		
DESCRIPTION	The SLPUnescape() function processes the input string in <i>pclnbuf</i> and unescapes any SLP reserved characters. If the <i>isTag</i> parameter is SLPTrue, then look for bad tag characters and signal an error if any are found with the SLP_PARSE_ERROR code. No transformation is performed if the input string is an opaque. The results are put into a buffer allocated by the API library and returned in the <i>ppcOutBuf</i> parameter. This buffer should be deallocated using SLPFree(3SLP) when the memory is no longer needed.		
PARAMETERS	pcInBuf	Pointer to the input buf	fer to process for escape characters.
	ppcOutBuf		the output buffer with the SLP reserved st be freed using SLPFree(3SLP) when the eded.
	isTag	When true, the input bu	ffer is checked for bad tag characters.
ERRORS	This function or its callback may return any SLP error code. See the ERRORS section in slp_api(3SLP).		
EXAMPLES	EXAMPLE 1 Using SLPUnescape()		
	The following example decodes the representation for ",tag,":		
	char* pcOutBuf; SLPError err;		
	err = SLPUnescape("\\2c tag\\2c", &pcOutbuf, SLP_TRUE);		
ENVIRONMENT	SLP_CONF_FILE When set, use this file for configuration.		
VARIABLES ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTE		ATTRIBUTE VALUE
	Availability		SUNWslpu
			-
SEE ALSO	<pre>slpd(1M),SLPFree(3SLP), slp_api(3SLP), slp.conf(4), slpd.reg(4), attributes(5)</pre>		
	System Administration Guide: Resource Management and Network Services		
	Guttman, E., Perkins, C., Veizades, J., and Day, M., <i>RFC 2608, Service Location Protocol, Version 2</i> , The Internet Society, June 1999.		

SLPUnescape(3SLP)

**NAME** | socket – create an endpoint for communication **SYNOPSIS** cc [ flag ... ] file ... -lsocket -lnsl [ library ... ] #include <sys/types.h> #include <sys/socket.h> int socket(int domain, int type, int protocol); DESCRIPTION socket () creates an endpoint for communication and returns a descriptor. The *domain* 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 <sys/socket.h>. There must be an entry in the netconfig(4) file for at least each protocol family and type required. If *protocol* has been specified, but no exact match for the tuplet 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: PF UNIX UNIX system internal protocols PF INET Internet Protocol Version 4 (IPv4) PF INET6 Internet Protocol Version 6 (IPv6) PF NCA Network Cache and Accelerator (NCA) protocols The socket has the indicated type, which specifies the communication semantics. Currently defined types are: SOCK\_STREAM SOCK DGRAM SOCK RAW SOCK SEQPACKET SOCK RDM 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 the superuser, and SOCK\_RDM, for which no implementation currently exists, are not described here.

*protocol* 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

### socket(3SOCKET)

SOCKET(SSOCKET)			
	"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 <i>connected</i> 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 a 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 <i>options</i> . These options are defined in the file <sys socket.h="">. setsockopt(3SOCKET) and getsockopt(3SOCKET) are used to set and get options, respectively.</sys>		
<b>RETURN VALUES</b>	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.	
	I		

576 man pages section 3: Networking Library Functions • Last Revised 5 Feb 2001

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

SEE ALSO nca(1), close(2), fcntl(2), ioctl(2), read(2), write(2), accept(3SOCKET), bind(3SOCKET), connect(3SOCKET), getsockname(3SOCKET), getsockopt(3SOCKET), listen(3SOCKET), recv(3SOCKET), setsockopt(3SOCKET), send(3SOCKET), shutdown(3SOCKET), socketpair(3SOCKET), attributes(5), in(3HEAD), socket(3HEAD)

# socket(3XNET)

NAME	socket – create an endpoint for communication		
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	int <b>socket</b> (int	domain, int type, int protocol);	
DESCRIPTION	The socket() function creates an unbound socket in a communications domain, and returns a file descriptor that can be used in later function calls that operate on sockets.		
	The function takes	The function takes the following arguments:	
	domain	Specifies the communications domain in which a socket is to be created.	
	type	Specifies the type of socket to be created.	
	protocol	Specifies a particular protocol to be used with the socket. Specifying a <i>protocol</i> of 0 causes socket() to use an unspecified default protocol appropriate for the requested socket type.	
	The <i>domain</i> argument specifies the address family used in the communications domain. The address families supported by the system are implementation-dependent.		
	The <sys socket.h=""> header defines at least the following values for the <i>domain</i> argument:</sys>		
	AF_UNIX	File system pathnames.	
	AF_INET	Internet Protocol version 4 (IPv4) address.	
	AF_INET6	Internet Protocol version 6 (IPv6) address.	
	The <i>type</i> argument specifies the socket type, which determines the semantics of communication over the socket. The socket types supported by the system are implementation-dependent. Possible socket types include:		
	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.	
	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.	
	SOCK_SEQPACK	ET Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.	

			SOCKET(SAINET)
			t specify a protocol that is supported by the
	The process may need to have appropriate privileges to use the socket() function or to create some sockets.		
USAGE	The documentation for specific address families specify which protocols each address family supports. The documentation for specific protocols specify which socket types each protocol supports.		
	The application can determine socket with <i>domain</i> set to the p		s family is supported by trying to create a lestion.
RETURN VALUES			turns a nonnegative integer, the socket file ned and errno is set to indicate the error.
ERRORS	The socket () function will	fail if:	
	EAFNOSUPPORT	The implem address fam	entation does not support the specified illy.
	EMFILE	No more file	e descriptors are available for this process.
	ENFILE	No more file	e descriptors are available for the system.
	EPROTONOSUPPORT		l is not supported by the address family, or is not supported by the implementation.
	EPROTOTYPE	The socket t	ype is not supported by the protocol.
	The socket () function may fail if:		
	EACCES The process does not have appropriate privileges.		
	ENOBUFS	Insufficient perform the	resources were available in the system to operation.
	ENOMEM	Insufficient	memory was available to fulfill the request.
	ENOSR		insufficient STREAMS resources available ation to complete.
ATTRIBUTES	See attributes(5) for descr	iptions of the	following attributes:
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT-Level		MT-Safe
SEE ALSO	getsockopt(3XNET), liste	en(3XNET), r JET), sendms	(3XNET), getsockname(3XNET), ecv(3XNET), recvfrom(3XNET), g(3XNET), setsockopt(3XNET), , attributes(5)

# socketpair(3SOCKET)

NAME	socketpair – create a pair of connected sock	kets	
SYNOPSIS	<pre>cc [ flag ] filelsocket -lnsl #include <sys types.h=""> #include <sys socket.h=""></sys></sys></pre>	[ library ]	
	<pre>int socketpair(int domain, int type, int protocol, int sv[2]);</pre>		
DESCRIPTION	The socketpair() library call creates an unnamed pair of connected sockets in the specified address family $d$ , of the specified type, and using the optionally specified <i>protocol</i> . The descriptors used in referencing the new sockets are returned in $sv[0]$ and $sv[1]$ . The two sockets are indistinguishable.		
<b>RETURN VALUES</b>	socketpair() returns -1 on failure, and	0 on success.	
ERRORS	The call succeeds unless:		
		The specified address family is not supported on this machine.	
		Too many descriptors are in use by this process.	
		There was insufficient user memory for the operation to complete.	
		There were insufficient STREAMS resources for the operation to complete.	
		The specified protocol does not support creation of socket pairs.	
		The specified protocol is not supported on this machine.	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Safe	
SEE ALSO	<pre>pipe(2), read(2), write(2), attributes</pre>	(5), socket(3HEAD)	
NOTES	This call is currently implemented only for	the AF_UNIX address family.	

NAME	socketpair – create	e a pair of co	onnected sockets
SYNOPSIS	<pre>cc [ flag ] filelxnet [ library ] #include <sys socket.h=""></sys></pre>		
	int socketpain	<b>r</b> (int <i>domu</i>	in, int type, int protocol, int socket_vector[2]);
DESCRIPTION	The socketpair() function creates an unbound pair of connected sockets in a specified <i>domain</i> , of a specified type, under the protocol optionally specified by the <i>protocol</i> argument. The two sockets are identical. The file descriptors used in referencing the created sockets are returned in <i>socket_vector</i> 0 and <i>socket_vector</i> 1.		d type, under the protocol optionally specified by the ckets are identical. The file descriptors used in
	<i>domain</i> Specifies the communications domain in which the sockets are to be created.		
	type	Specifies t	he type of sockets to be created.
	protocolSpecifies a particular protocol to be used with the sockets.Specifying a protocol of 0 causes socketpair() to use an unspecified default protocol appropriate for the requested socket type.		
	socket_vector	<i>ector</i> Specifies a 2-integer array to hold the file descriptors of the created socket pair.	
	The <i>type</i> argument specifies the socket type, which determines the semantics of communications over the socket. The socket types supported by the system are implementation-dependent. Possible socket types include:		
	SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.		connection-mode byte streams, and may provide a
	SOCK_DGRAM		Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.
	SOCK_SEQPACK	ET	Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.
	If the <i>protocol</i> argument is non-zero, it must specify a protocol that is supported by the address family. The protocols supported by the system are implementation-dependent.		
	The process may r function or to crea		e appropriate privileges to use the socketpair() kets.
	1		

socketpair(3XNET)		
USAGE	The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.	
	The socketpair() function not be supported for other do	is used primarily with UNIX domain sockets and need mains.
RETURN VALUES	Upon successful completion, this function returns 0. Otherwise, -1 is returned and errno is set to indicate the error.	
ERRORS	The socketpair() function	will fail if:
	EAFNOSUPPORT	The implementation does not support the specified address family.
	EMFILE	No more file descriptors are available for this process.
	ENFILE	No more file descriptors are available for the system.
	EOPNOTSUPP	The specified protocol does not permit creation of socket pairs.
	EPROTONOSUPPORT	The protocol is not supported by the address family, or the protocol is not supported by the implementation.
	EPROTOTYPE	The socket type is not supported by the protocol.
	The socketpair() function	may fail if:
	EACCES	The process does not have appropriate privileges.
	ENOBUFS	Insufficient resources were available in the system to perform the operation.
	ENOMEM	Insufficient memory was available to fulfill the request.
	ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SEE ALSO** socket(3XNET), attributes(5)

582 man pages section 3: Networking Library Functions • Last Revised 8 May 1998

		spra)(00000121)
NAME	spray – scatter data in order to te	est the network
SYNOPSIS	<pre>cc [ flag ] filelsocke #include <rpcsvc spray.h=""></rpcsvc></pre>	t -lnsl [ <i>library</i> ]
	bool_t xdr_sprayarr(XDR ;	*xdrs, sprayarr *objp);
	bool_t xdr_spraycumul(XD)	R * <i>xdrs</i> , spraycumul * <i>objp</i> );
DESCRIPTION	The spray program sends packet machine.	ts to a given machine to test communications with that
	the generic remote procedure cal rpc_clnt_calls(3NSL). The p	nction interface, per se, but it can be accessed using lling interface clnt_call(). See program sends a packet to the called host. The host ket. The program counts the number of n that count.
	The spray program currently sup called in the order given:	oports the following procedures, which should be
	SPRAYPROC_CLEAR T	his procedure clears the counter.
	SPRAYPROC_SPRAY T	his procedure sends the packet.
	_	his procedure returns the count and the amount of me since the last SPRAYPROC_CLEAR.
EXAMPLES	EXAMPLE 1 Using spray()	
	The following code fragment der	monstrates how the spray program is used:
	<pre>#include <rpc rpc.h=""> #include <rpcsvc spray.h=""> spraycumul spray_result. </rpcsvc></rpc></pre>	;
	<pre>sprayarr spray_data; char buf[100]; /* arbitrary data */ int loop = 1000; CLIENT *clnt; struct timeval timeout0 = {0, 0}; struct timeval timeout25 = {25, 0};</pre>	
	<pre>spray_data.sprayarr_len = spray_data.sprayarr_val = 1 clnt = clnt_create("somehow if (clnt == (CLIENT *)NULL)</pre>	ouf; st", SPRAYPROG, SPRAYVERS, "netpath"); ) {
	<pre>/ if (clnt_call(clnt, SPRAYPROC_CLEAR,     xdr_void, NULL, xdr_void, NULL, timeout25)) {         /* handle this error */ } while (loop- &gt; 0) {         if (clnt_call(clnt, SPRAYPROC_SPRAY,             xdr_sprayarr, &amp;spray_data, xdr_void, NULL, timeout0)) {             /* handle this error */         </pre>	

```
spray(3SOCKET)
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SEE ALSO** | spray(1M), rpc\_clnt\_calls(3NSL), attributes(5)

**NOTES** This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

A spray program is not useful as a networking benchmark as it uses unreliable connectionless transports, for example, udp. It can report a large number of packets dropped, when the drops were caused by the program sending packets faster than they can be buffered locally, that is, before the packets get to the network medium.

NAME	AME   t_accept – accept a connection request	
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>	
	<pre>int t_accept(int fd, int resfd, const struct t_call *call);</pre>	
DESCRIPTION	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, tiuser.h, must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.	
	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 t_call structure which contains the following members:	
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;In call, addr is the protocol address of the calling transport user, opt indicates any options associated with the connection, udata points to any user data to be returned to the caller, and sequence is the value returned by t_listen(3NSL) that uniquely associates the response with a previously received connection indication. Th address of the caller, addr may be null (length zero). Where addr is not null then it may optionally be checked by XTI.</pre>	
	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==fd</i> ), the user must have responded to any previous connection indications received on that transport endpoint by means of t_accept() or t_snddis(3NSL). Otherwise, t_accept() will fail and set t_errno to TINDOUT.	
	If a different transport endpoint is specified ( $resfd!=fd$ ), then the user may or may not choose to bind the endpoint before the t_accept() is issued. If the endpoint is not bound prior to the t_accept(), the endpoint must be in the T_UNBND state before the t_accept() 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 t_accept() is issued.	
	Responding endpoints should be supplied to $t_accept()$ in the state $T_UNBND$ .	
	The call to t_accept() may fail with t_errno 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.	
	Networking Library Functions 585	

t_accept(3NSL)		
	The <i>udata</i> 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 <i>connect</i> field of the <i>info</i> argument of t_open(3NSL) or t_getinfo(3NSL). If the <i>len</i> field of <i>udata</i> is zero, no data will be sent to the caller. All the <i>maxlen</i> fields are meaningless.	
		ate any option ( $call \rightarrow opt.len = 0$ ) the connection shall be uses currently set for the responding endpoint <i>resfd</i> .
<b>RETURN VALUES</b>	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.	
VALID STATES	fd: T_INCON	
	resfd (fd!=resfd): T_II	DLE, 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.
	TBADOPT	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 $fd==resfd$ 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.

	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 (t_errno).
	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.
TLI COMPATIBILITY	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.	
Interface Header	The XTI interfaces use the hea header. They should use the h	nder file, xti.h. TLI interfaces should <i>not</i> use this neader:
	<pre>#include <tiuser.h></tiuser.h></pre>	
Error Description Values	The t_errno values that can interface are:	be set by the XTI interface and cannot be set by the TLI
	TPROTO	
	TINDOUT	
	TPROVMISMATCH	
	TRESADDR	
	TRESQLEN	
Option Buffer		n opt buffer is dictated by the transport provider. Unlike rface does not specify the buffer format.
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:

t\_accept(3NSL)

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT Level	Safe
SEE ALSO	t_connect(3NSL), t_getinfo(3NSL), t_ t_open(3NSL), t_optmgmt(3NSL), t_rcv attributes(5)	
WARNINGS	There may be transport provider-specific re	estrictions on address binding.
	Some transport providers do not differentia connection itself. If the connection has alrea return of t_listen(3NSL), t_accept() w transport endpoint specified by <i>resfd</i> .	dy been established after a successful

NAME	t_alloc – allocate a library structure	
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>	
	<pre>void *t_alloc(int fd, int struct_type, int fields);</pre>	
DESCRIPTION	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, a different header file, tiuser.h, must be used. Refer to the section, TLI COMPATIBILITY, for a description of differences between the two interfaces. The t_alloc() function dynamically allocates memory for the various transport function argument structures as specified below. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.	
	The structure to al	locate is specified by <i>struct_type</i> and must be one of the following:
	T_BINDstructt_bindT_CALLstructt_callT_OPTMGMTstructt_optmgmtT_DISstructt_disconT_UNITDATAstructt_unitdataT_UDERCORstructt_infowhere each of these structures may subsequently be used as an argument to one ormore transport functions.Each of the above structures, except T_INFO, contains at least one field of typestruct netbuf. For each field of this type, the user may specify that the buffer forthat field should be allocated as well. The length of the buffer allocated will be equalto or greater than the appropriate size as returned in the <i>info</i> argument oft_open(3NSL) or t_getinfo(3NSL). The relevant fields of the <i>info</i> argument aredescribed in the following list. The <i>fields</i> argument specifies which buffers to allocate,where the argument is the bitwise-or of any of the following:	
	T_ADDR	The <i>addr</i> field of the t_bind, t_call, t_unitdata or t_uderr structures.
T_OPT T_UDATA	T_OPT	The <i>opt</i> field of the t_optmgmt, t_call, t_unitdata or t_uderr structures.
	T_UDATA	The <i>udata</i> field of the t_call, t_discon or t_unitdata structures.
	T_ALL	All relevant fields of the given structure. Fields which are not supported by the transport provider specified by <i>fd</i> will not be allocated.
		field specified in <i>fields</i> , t_alloc() will allocate memory for the with the field, and initialize the <i>len</i> field to zero and the <i>buf</i> pointer

Networking Library Functions 589

t\_alloc(3NSL)

	ignored. Since the information that is t_getinfo(3NSL allocated structure allocated, <i>fd</i> may b be accessed. If the t_alloc() will b setting t_errno t t_getinfo(3NSL T_INFINITE, the field not specified set to zero. See t_	ccordingly. Irrelevant or unknown values passed in fields are length of the buffer allocated will be based on the same size is returned to the user on a call to t_open(3NSL) and L), fd must refer to the transport endpoint through which the newly e will be passed. In the case where a T_INFO structure is to be be set to any value. In this way the appropriate size information can size value associated with any specified field is T_INVALID, be unable to determine the size of the buffer to allocate and will fail, to TSYSERR and errno to EINVAL. See t_open(3NSL) or L). If the size value associated with any specified field is en the behavior of t_alloc() is implementation-defined. For any in fields, buf will be set to the null pointer and <i>len</i> and <i>maxlen</i> will be open(3NSL) or t_getinfo(3NSL).
	The pointer returned if the allocation succeeds is suitably aligned so that it can be assigned to a pointer to any type of object and then used to access such an object or array of such objects in the space allocated.	
		) to allocate structures will help ensure the compatibility of user ture releases of the transport interface functions.
RETURN VALUES	On successful completion, t_alloc() returns a pointer to the newly allocated structure. On failure, a null pointer is returned.	
VALID STATES	ALL - apart from T_UNINIT	
ERRORS	On failure, t_errno is set to one of the following:	
	TBADF	struct_type is other than T_INFO and the specified file descriptor does not refer to a transport endpoint.
	TNOSTRUCTYPE	Unsupported <i>struct_type</i> requested. This can include a request for a structure type which is inconsistent with the transport provider type specified, that is, connection-mode or connectionless-mode.
	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.
TLI COMPATIBILITY	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.	
Interface Header		use the header file, xti.h. TLI interfaces should <i>not</i> use this all use the header:
	#include <tius< th=""><th>er.h&gt;</th></tius<>	er.h>

# t\_alloc(3NSL)

Error Description Values	The t_errno values that can be set by the interface are:	XTI interface and cannot be set by the TLI
	TPROTO	
	TNOSTRUCTYPE	
Special Buffer Sizes	Assume that the value associated with any returned by t_open() or t_getinfo() ) underlying service provider can support a lt_alloc() will allocate a buffer with the chandled as described in the next paragraph	that describes buffer limits is –1. Then the puffer of unlimited size. If this is the case, lefault size 1024 bytes, which may be
	If the underlying service provider supports structure (see t_connect(3NSL)), t_allo If a larger size buffer is required, it will nee- memory allocation routine such as malloc netbuf data structure can then be updated 1024 byte buffer originally allocated by t_a	c() will return a buffer of size 1024 bytes. d to be allocated separately using a (3C). The buf and maxlen fields of the with the address of the new buffer and the
	Assume that the value associated with any returned by t_open() or t_getinfo() t_alloc() will set the buffer pointer to NU then will return success (see t_open(3NSL))	) that describes nbuffer limits is $-2$ . Then ILL and the buffer maximum size to 0, and
	See attributes(5) for descriptions of the following attributes:	
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
ATTRIBUTES	ATTRIBUTE TYPE	following attributes:           ATTRIBUTE VALUE
ATTRIBUTES		
SEE ALSO		ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe
	ATTRIBUTE TYPE MT Level free(3C), malloc(3C), t_connect(3NSL)	ATTRIBUTE VALUE Safe

## t\_bind(3NSL)

NAME	t_bind – bind an address to a transport endpoint
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_bind(int fd, const struct t_bind *req, struct t_bind *ret);</pre>
DESCRIPTION	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 tiuser.hheader file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.
	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 enqueuing 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.
	The <i>req</i> and <i>ret</i> arguments point to a t_bind structure containing the following members:
	struct netbuf addr; unsigned qlen;
	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.
	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.
	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.
	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

in the negotiated value of <i>qlen</i> . It is valid to	set <i>req</i> and <i>ret</i> to the null pointer for the
same call, in which case the provider choos	es the address to bind to the transport
endpoint and does not return that information	tion 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 at unbind(3NSL) ort 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** Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t\_errno is set to indicate an error.

VALID STATES T\_UNBND

**ERRORS** On failure, t\_errno is set to one of the following:

t\_bind(3NSL)

_ 、 ,	TACCES	The user does not have	permission to use the specified address.
	TADDRBUSY	The requested address i	
	TBADADDR	-	address was in an incorrect format or
	TBADF	The specified file descri	ptor does not refer to a transport endpoint.
	TBUFOVFLW	greater than 0 but not s	lowed for an incoming argument ( <i>maxlen</i> ) is ufficient to store the value of that argument. Il change to T_IDLE and the information to e discarded.
	TOUTSTATE	The communications er states in which a call to	adpoint referenced by $fd$ is not in one of the this function is valid.
	TNOADDR	The transport provider	could not allocate an address.
	TPROTO		t a communication problem has been nd the transport provider for which there is ror (t_errno).
	TSYSERR	A system error has occu	rred during execution of this function.
TLI COMPATIBILITY	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.		
Interface Header		use the header file, xti. Ild use the header:	h. TLI interfaces should <i>not</i> use this
	#include <tius< th=""><th>ser.h&gt;</th><th></th></tius<>	ser.h>	
Address Bound			and <i>ret</i> to determine whether the transport different address than that requested.
Error Description Values	The t_errno valucannot be set by the		USY can be set by the XTI interface but
	XTI counterpart is		rn under different circumstances than its turned even when the maxlen field of the
ATTRIBUTES	See attributes(	(5) for descriptions of the	following attributes:
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe

594 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

SEE ALSO | 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\_close(3NSL)

NAME	t_close – close a tra	ansport endpoint
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>	
	int <b>t_close</b> (in	tfd);
DESCRIPTION	represents the futu supported for com XTI routine, the t	t of the XTI interfaces which evolved from the TLI interfaces. XTI are evolution of these interfaces. However, TLI interfaces are apatibility. When using a TLI routine that has the same name as an iuser.h header file must be used. Refer to the TLI section for a description of differences between the two interfaces.
	the transport endp	unction informs the transport provider that the user is finished with point specified by $fd$ , and frees any local library resources associated . In addition, t_close() closes the file associated with the t.
	t_getstate(3NS may be called from library resources a close(2) will be in process or in anoth connection that ma	Lose () should be called from the T_UNBND state. See SL). However, this function does not check state information, so it in any state to close a transport endpoint. If this occurs, the local associated with the endpoint will be freed automatically. In addition, ssued for that file descriptor; if there are no other descriptors in this her process which references the communication endpoint, any ay be associated with that endpoint is broken. The connection may n orderly or abortive manner.
	data not yet receiv	ued on a connection endpoint may cause data previously sent, or red, to be lost. It is the responsibility of the transport user to ensure ed by the remote peer.
<b>RETURN VALUES</b>		ompletion, a value of 0 is returned. Otherwise, a value of –1 is _errno is set to indicate an error.
VALID STATES	T_UNBND	
ERRORS	On failure, t_err	no is set to the following:
	TBADF	The specified file descriptor does not refer to a transport endpoint.
	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.
TLI COMPATIBILITY		nterface definitions have common names but use different header er semantic differences between the two interfaces are described in low.
Interface Header	The XTI interfaces header. They shou	use the header file, xti.h. TLI interfaces should <i>not</i> use this ld use the header:

596 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

#include <tiuser, h<="" td="">         Error Description Interface is::::::::::::::::::::::::::::::::::::</tiuser,>			t_close(3NSL)
TROTO         ATTRIBUTES         See attributes(5) for descriptions of the following attributes:         Image: see attributes (5) for descriptions of the following attributes (5)         SEE ALSO         close(2), t_getstate(3NSL), t_open(3NSL), t_unbind(3NSL), attributes(5)		<pre>#include <tiuser.h></tiuser.h></pre>	
ATTRIBUTES       See attributes(5) for descriptions of the following attributes: <ul> <li></li></ul>	Error Description Values	The t_errno value that can be set by the X interface is:	(TI interface and cannot be set by the TLI
ATTRIBUTE TYPE     ATTRIBUTE VALUE       MT Level     Safe       SEE ALSO     close(2), t_getstate(3NSL), t_open(3NSL), t_unbind(3NSL), attributes(5)		TPROTO	
MT Level     Safe       SEE ALSO     close(2), t_getstate(3NSL), t_open(3NSL), t_unbind(3NSL), attributes(5)	ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
MT Level     Safe       SEE ALSO     close(2), t_getstate(3NSL), t_open(3NSL), t_unbind(3NSL), attributes(5)		ATTRIBUTE TYPE	
Ntworking Library Function         577	SEE ALSO	close(2), t_getstate(3NSL), t_open(3N	NSL),t_unbind(3NSL),attributes(5)
517			
<b>177</b>			
197			
Starting Library Function       57			
Networking Library Function <b>57</b> 7			
Networking Library Functions 597			
Networking Library Functions 577			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
Networking Library Functions 597			
с , с ,			Networking Library Functions 597

t_connect(	(3NSL)
------------	--------

NAME	t_connect – establish a connection with another transport user
SYNOPSIS	#include <xti.h></xti.h>
51101315	<pre>int t_connect(int fd, const struct t_call *sndcall, struct t_call     *rcvcall);</pre>
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces. This function enables a transport user to request a connection to the specified destination transport user.
	This function can only be issued in the T_IDLE state. The parameter <i>fd</i> identifies the local transport endpoint where communication will be established, while <i>sndcall</i> and <i>rcvcall</i> point to a t_call structure which contains the following members:
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre>
	The parameter <i>sndcall</i> specifies information needed by the transport provider to establish a connection and <i>rcvcall</i> specifies information that is associated with the newly established connection.
	In <i>sndcall, addr</i> specifies the protocol address of the destination transport user, <i>opt</i> presents any protocol-specific information that might be needed by the transport provider, <i>udata</i> points to optional user data that may be passed to the destination transport user during connection establishment, and <i>sequence</i> has no meaning for this function.
	On return, in <i>rcvcall</i> , <i>addr</i> contains the protocol address associated with the responding transport endpoint, <i>opt</i> represents any protocol-specific information associated with the connection, <i>udata</i> points to optional user data that may be returned by the destination transport user during connection establishment, and <i>sequence</i> has no meaning for this function.
	The <i>opt</i> argument permits users to define the options that may be passed to the transport provider. The user may choose not to negotiate protocol options 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.
	If used, <i>sndcall</i> $\rightarrow$ <i>opt.buf</i> must point to a buffer with the corresponding options, and <i>sndcall</i> $\rightarrow$ <i>opt.len</i> must specify its length. The <i>maxlen</i> and <i>buf</i> fields of the netbuf structure pointed by <i>rcvcall</i> $\rightarrow$ <i>addr</i> and <i>rcvcall</i> $\rightarrow$ <i>opt</i> must be set before the call.

	The <i>udata</i> argument enables the caller to pass user data to the destination transport user and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned in the <i>connect</i> field of the <i>info</i> argument of t_open(3NSL) or t_getinfo(3NSL). If the <i>len</i> of <i>udata</i> is zero in <i>sndcall</i> , no data will be sent to the destination transport user.		
	On return, the <i>addr</i> , <i>opt</i> and <i>udata</i> fields of <i>rcvcall</i> will be updated to reflect values associated with the connection. Thus, the <i>maxlen</i> field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, <i>maxlen</i> can be set to zero, in which case no information to this specific argument is given to the user on the return from t_connect(). If maxlen is greater than zero and less than the length of the value, t_connect() fails with t_errno set to TBUFOVFLW. If <i>rcvcall</i> is set to NULL, no information at all is returned.		
	By default, t_connect() executes in synchronous mode, and will wait for the destination user's response before returning control to the local user. A successful return (that is, return value of zero) indicates that the requested connection has been established. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_connect() executes in asynchronous mode. In this case, the call will not wait for the remote user's response, but will return control immediately to the local user and return -1 with t_errno set to TNODATA to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connection request to the destination transport user. The t_rcvconnect(3NSL) function is used in conjunction with t_connect() to determine the status of the requested connection.		
	state of the corresp either t_rcvconn asynchronous t_c	<pre>bus t_connect() call is interrupted by the arrival of a signal, the bonding transport endpoint is T_OUTCON, allowing a further call to ecct(3NSL), t_rcvdis(3NSL) or t_snddis(3NSL). When an bonnect() call is interrupted by the arrival of a signal, the state of transport endpoint is T_IDLE.</pre>	
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE		
ERRORS	On failure, t_errno is set to one of the following:		
	TACCES	The user does not have permission to use the specified address or options.	
	TADDRBUSY	This transport provider does not support multiple connections with the same local and remote addresses. This error indicates that a connection already exists.	
	TBADADDR	The specified protocol address was in an incorrect format or contained illegal information.	

t\_connect(3NSL)

, ,		
	TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.
	TBADF	The specified file descriptor does not refer to a transport endpoint.
	TBADOPT	The specified protocol options were in an incorrect format or contained illegal information.
	TBUFOVFLW	The number of bytes allocated for an incoming argument ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to T_DATAXFER, and the information to be returned in <i>rcvcall</i> is discarded.
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
	TNODATA	O_NONBLOCK was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user.
	TNOTSUPPORT	This function is not supported by the underlying transport provider.
	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).
	TSYSERR	A system error has occurred during execution of this function.
TLI COMPATIBILITY		nterface definitions have common names but use different header er semantic differences between the two interfaces are described in low.
Interface Header	The XTI interfaces header. They shou	use the header file, xti.h. TLI interfaces should <i>not</i> use this ld use the header:
	#include <tiuser.]< th=""><th>h&gt;</th></tiuser.]<>	h>
Error Description Values	The TPROTO and T by the TLI interfac	CADDRBUSY t_errno values can be set by the XTI interface but not e.
	XTI counterpart is	that this routine can return under different circumstances than its TBUFOVFLW. It can be returned even when the maxlen field of the fer has been set to zero.
<b>Option Buffers</b>		options in an opt buffer is dictated by the transport provider. Unlike he TLI interface does not fix the buffer format.

#### t\_connect(3NSL)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

# SEE ALSO fcntl(2), t\_accept(3NSL), t\_alloc(3NSL), t\_getinfo(3NSL), t\_listen(3NSL), t\_open(3NSL), t\_optmgmt(3NSL), t\_rcvconnect(3NSL), t\_rcvdis(3NSL), t\_snddis(3NSL), attributes

t_errno(3NSL)	
NAME	t_errno – XTI error return value
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
DESCRIPTION	This error return value 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 interface that has the same name as an XTI interfaces, a different headerfile, <tiuser.h>, must be used. Refer the the TLI COMPATIBILITY section for a description of differences between the two interfaces.</tiuser.h>
	t_errno is used by XTI functions to return error values.
	XTI functions provide an error number in t_errno which has type <i>int</i> and is defined in <xti.h>. The value of t_errno will be defined only after a call to a XTI function for which it is explicitly stated to be set and until it is changed by the next XTI function call. The value of t_errno should only be examined when it is indicated to be valid by a function's return value. Programs should obtain the definition of t_errno by the inclusion of <xti.h>. The practice of defining t_errno in program as extern int t_errno is obsolescent. No XTI function sets t_errno to 0 to indicate an error.</xti.h></xti.h>
	It is unspecified whether t_errno is a macro or an identifier with external linkage. It represents a modifiable lvalue of type <i>int</i> . If a macro definition is suppressed in order to access an actual object or a program defines an identifier with name <i>t_errno</i> , the behavior is undefined.
	The symbolic values stored in t_errno by an XTI function are defined in the ERRORS sections in all relevant XTI function definition pages.
TLI COMPATIBILITY	t_errno is also used by TLI functions to return error values.
COMIATIBILITT	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.
Interface Header	The XTI interfaces use the header file, <xti.h>. TLI interfaces should <i>not</i> use this header. They should use the header:</xti.h>
	<pre>#include <tiuser.h></tiuser.h></pre>
Error Description Values	The t_errno values that can be set by the XTI interface but cannot be set by the TLI interface are:
	TNOSTRUCTYPE TBADNAME TBADQLEN TADDRBUSY

602 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

## t\_errno(3NSL)

TINDOUT	
TPROVMISMATCH	
TRESADDR	
TQFULL	
TPROTO	

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO attributes(5)

# t\_error(3NSL)

NAME	t_error – produce error message
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_error(const char *errmsg);</pre>
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.
	The t_error() function produces a message on the standard error output which describes the last error encountered during a call to a transport function. The argument string <i>errmsg</i> is a user-supplied error message that gives context to the error.
	The error message is written as follows: first (if <i>errmsg</i> is not a null pointer and the character pointed to be <i>errmsg</i> is not the null character) the string pointed to by <i>errmsg</i> followed by a colon and a space; then a standard error message string for the current error defined in t_errno. If t_errno has a value different from TSYSERR, the standard error message string is followed by a newline character. If, however, t_errno is equal to TSYSERR, the t_errno string is followed by the standard error message string for the current error defined in errno followed by a newline.
	The language for error message strings written by t_error() is that of the current locale. If it is English, the error message string describing the value in t_errno may be derived from the comments following the t_errno codes defined in xti.h. The contents of the error message strings describing the value in errno are the same as those returned by the strerror(3C) function with an argument of errno.
	The error number, t_errno, is only set when an error occurs and it is not cleared on successful calls.
EXAMPLES	If a t_connect(3NSL) function fails on transport endpoint <i>fd2</i> because a bad address was given, the following call might follow the failure:
	$\texttt{t\_error("t\_connect failed on fd2")}; The diagnostic message to be printed would look like:$
	t_connect failed on fd2: incorrect addr formatwhere <i>incorrect addr format</i> identifies the specific error that occurred, and <i>t_connect failed on fd2</i> tells the user which function failed on which transport endpoint.
<b>RETURN VALUES</b>	Upon completion, a value of 0 is returned.
VALID STATES	All - apart from T_UNINIT
ERRORS	No errors are defined for the t_error() function.

# t\_error(3NSL)

TLI COMPATIBILITY	The XTI and TLI interface definitions have files. This, and other semantic differences be the subsections below.		
Interface Header	The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:		
	<pre>#include <tiuser.h></tiuser.h></pre>		
Error Description Values	The t_errno value that can be set by the X interface is:	(TI interface and cannot be set by the TLI	
	TPROTO		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT Level	Safe	
SEE ALSO	MT Level t_errno(3NSL)strerror(3C), attribut		
SEE ALSO			

# t\_free(3NSL)

NAME	t_free – free a library structure		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_free(void *ptr, int struct_type);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
		memory previously allocated by t_alloc(3NSL). This the specified structure, and will also free memory for sture.	
		e of the seven structure types described for <i>ppe</i> identifies the type of that structure which must be	
	T_BIND struct t_bind T_CALL struct t_call T_OPTMGMT struct t_optmgmt T_DIS struct t_discon T_UNITDATA struct t_unitdata T_UDERROR struct t_uderr T_INFO struct t_info		
	where each of these structures is used as an argument to one or more transport functions.		
	The function t_free() will check the <i>addr</i> , <i>opt</i> and <i>udata</i> fields of the given structure, as appropriate, and free the buffers pointed to by the <i>buf</i> field of the netbuf structure. If <i>buf</i> is a null pointer, t_free() will not attempt to free memory. After all buffers are freed, t_free() will free the memory associated with the structure pointed to by <i>ptr</i> .		
	Undefined results will occur if <i>ptr</i> or any of the <i>buf</i> pointers points to a block of memory that was not previously allocated by t_alloc(3NSL).		
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	ALL - apart from T_UNINIT.		
ERRORS	On failure, t_errno is set to the following:		
	TNOSTRUCTYPE	Unsupported <i>struct_type</i> requested.	
	TPROTOThis error indicates that a communication problem had been detected between XTI and the transport provides for which there is no other suitable XTI error ( <i>t_errno</i> )		

t\_free(3NSL)

	TSYSERR	A system err function.	or has occurred during execution of this	
TLI COMPATIBILITY	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.			
Interface Header	The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:			
	<pre>#include <tiuser.h></tiuser.h></pre>			
Error Description Values	The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:			
	TPROTO			
ATTRIBUTES	See attributes(5) for descri	ptions of the	following attributes:	
	ATTRIBUTE TYPE		ATTRIBUTE VALUE	
	MT Level		Safe	
SEE ALSO	t_alloc(3NSL),attribute	s(5)		

# t\_getinfo(3NSL)

NAME	t_getinfo – get protocol-specific service information	
SYNOPSIS	#include <xti.h></xti.h>	
	int <b>t_getinfo</b> (ir	<pre>it fd, struct t_info *info);</pre>
represents the future evolution of these int supported for compatibility. When using a XTI routine, the tiuser.h header file mu		f the XTI interfaces which evolved from the TLI interfaces. XTI evolution of these interfaces. However, TLI interfaces are atibility. When using a TLI routine that has the same name as an ser.h header file must be used. Refer to the TLI ction for a description of differences between the two interfaces.
	and/or transport cor to return the same in precisely the same va	the current characteristics of the underlying transport protocol nection associated with file descriptor $fd$ . The <i>info</i> pointer is used formation returned by t_open(3NSL), although not necessarily alues. This function enables a transport user to access this any phase of communication.
	This argument points	s to a t_info structure which contains the following members:
	t_scalar_t addr; t_scalar_t options; t_scalar_t tsdu; t_scalar_t etsdu;	<pre>/*max size in octets of the transport protocol address*/ /*max number of bytes of protocol-specific options */ /*max size in octets of a transport service data unit */ /*max size in octets of an expedited transport service*/ /*data unit (ETSDU) */</pre>
	t_scalar_t connect;	<pre>/*max number of octets allowed on connection */ /*establishment functions */</pre>
	t_scalar_t discon;	<pre>/*max number of octets of data allowed on t_snddis() */ /*and t_rcvdis() functions */</pre>
	<pre>t_scalar_t servtype; /*service type supported by the transport provider */ t_scalar_t flags; /*other info about the transport provider */ The values of the fields have the following meanings:</pre>	
	t: ti	A value greater than zero indicates the maximum size of a ransport protocol address and a value of T_INVALID (-2) specifies hat the transport provider does not provide user access to ransport protocol addresses.
	0	A value greater than zero indicates the maximum number of bytes f protocol-specific options supported by the provider, and a value f T_INVALID (-2) specifies that the transport provider does not upport user-settable options.
	ti s o v o o o o	A value greater than zero specifies the maximum size in octets of a ransport service data unit (TSDU); a value of T_NULL (zero) pecifies that the transport provider does not support the concept of TSDU, although it does support the sending of a datastream with no logical boundaries preserved across a connection; a value of T_INFINITE (-1) specifies that there is no limit on the size in octets of a TSDU; and a value of T_INVALID (-2) specifies that the ransfer of normal data is not supported by the transport provider.

etsdu	A value greater than zero specifies the maximum size in octets of an expedited transport service data unit (ETSDU); a value of T_NULL (zero) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of T_INFINITE (-1) specifies that there is no limit on the size (in octets) of an ETSDU; and a value of T_INVALID (-2) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers.
connect	A value greater than zero specifies the maximum number of octets that may be associated with connection establishment functions and a value of T_INVALID (-2) specifies that the transport provider does not allow data to be sent with connection establishment functions.
discon	If the T_ORDRELDATA bit in flags is clear, a value greater than zero specifies the maximum number of octets that may be associated with the t_snddis(3NSL) and t_rcvdis(3NSL) functions, and a value of T_INVALID (-2) specifies that the transport provider does not allow data to be sent with the abortive release functions. If the T_ORDRELDATA bit is set in flags, a value greater than zero specifies the maximum number of octets that may be associated with the t_sndreldata(),t_rcvreldata(),t_snddis(3NSL) and t_rcvdis(3NSL) functions.
servtype	This field specifies the service type supported by the transport provider, as described below.
flags	This is a bit field used to specify other information about the communications provider. If the T_ORDRELDATA bit is set, the communications provider supports sending user data with an orderly release. If the T_SENDZERO bit is set in flags, this indicates that the underlying transport provider supports the sending of zero-length TSDUs.
accessed to deterr Alternatively, the error will result if value of each field connection establi returned by t_ge	is concerned with protocol independence, the above sizes may be nine how large the buffers must be to hold each piece of information. t_alloc(3NSL) function may be used to allocate these buffers. An a transport user exceeds the allowed data size on any function. The d may change as a result of protocol option negotiation during shment (the t_optmgmt(3NSL) call has no effect on the values tinfo()). These values will only change from the values presented after the endpoint enters the T_DATAXFER state.

The *servtype* field of *info* specifies one of the following values on return:

t\_getinfo(3NSL)

=0 ( )		
	T_COTS	The transport provider supports a connection-mode service but does not support the optional orderly release facility.
	T_COTS_ORD	The transport provider supports a connection-mode service with the optional orderly release facility.
	T_CLTS	The transport provider supports a connectionless-mode service. For this service type, t_open(3NSL) will return T_INVALID (-1) for <i>etsdu</i> , <i>connect</i> and <i>discon</i> .
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an 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.
	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.
TLI COMPATIBILITY	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.	
Interface Header	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.< th=""><th>h&gt;</th></tiuser.<>	h>
Error Description Values	The t_errno value TPROTO can be set by the XTI interface but not by the TLI interface.	
The t_info Structure	For TLI , the t_info structure referenced by <i>info</i> lacks the following structure member:	
	t_scalar_t flags; was added to str	/* other info about the transport provider */This member uct t_info in the XTI interfaces.
	members, it signifi for a correspondin ETSDU (octet size)	1 is observed as the return value in various t_info structure ies that the transport provider can handle an infinite length buffer g attribute, such as address data, option data, TSDU (octet size), ), connection data, and disconnection data. The corresponding s are addr, options, tsdu, estdu, connect, and discon,
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:	

t\_getinfo(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

## SEE ALSO t\_alloc(3NSL), t\_open(3NSL), t\_optmgmt(3NSL), t\_rcvdis(3NSL), t\_snddis(3NSL), attributes(5)

# t\_getprotaddr(3NSL)

NAME	t_getprotaddr – get the protocol addresses		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	int <b>t_getprot</b> a *peeraddr);	<pre>addr(int fd, struct t_bind *boundaddr, struct t_bind</pre>	
DESCRIPTION	represents the futu supported for com XTI routine, the t	t of the XTI interfaces which evolved from the TLI interfaces. XTI are evolution of these interfaces. However, TLI interfaces are apatibility. When using a TLI routine that has the same name as an iuser.h header file must be used. Refer to the TLI section for a description of differences between the two interfaces.	
	The t_getprotaddr() function returns local and remote protocol addresses currently associated with the transport endpoint specified by <i>fd</i> . In <i>boundaddr</i> and <i>peeraddr</i> the user specifies <i>maxlen</i> , which is the maximum size (in bytes) of the address buffer, and <i>buf</i> which points to the buffer where the address is to be placed. On return, the <i>buf</i> field of <i>boundaddr</i> points to the address. If any, currently bound to <i>fd</i> , and the <i>len</i> field specifies the length of the address. If the transport endpoint is in the T_UNBND state, zero is returned in the <i>len</i> field of <i>boundaddr</i> . The <i>buf</i> field of <i>peeraddr</i> points to the address. If the transport endpoint is the length of the address. If the <i>I</i> field specifies the length of the address. If the <i>I</i> field specifies the length of the address. If the <i>I</i> field of <i>peeraddr</i> points to the address. If the <i>I</i> field of <i>peeraddr</i> points to the address. If the <i>I</i> field of <i>peeraddr</i> points to the address. If the transport endpoint is not in the <i>I</i> field of <i>peeraddr</i> . If the <i>maxlen</i> field of <i>boundaddr</i> or <i>peeraddr</i> is set to zero, no address is returned.		
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set 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.	
	TBUFOVFLW	The number of bytes allocated for an incoming argument ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the value of that argument.	
	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.	
TLI COMPATIBILITY ATTRIBUTES	In the TLI interface definition, no counterpart of this routine was defined.		
	See attributes(5) for descriptions of the following attributes:		

t\_getprotaddr(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SEE ALSO** t\_bind(3NSL), attributes(5)

Networking Library Functions 613

t_	getstate(3NSL)
----	----------------

NAME	t_getstate – get the current state		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_getstate(int fd);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
		() function returns the current state of the provider associated with point specified by <i>fd</i> .	
RETURN VALUES		pon successful completion. Otherwise, a value of –1 is returned and indicate an error. The current state is one of the following:	
	T_UNBND	Unbound.	
	T_IDLE	Idle.	
	T_OUTCON	Outgoing connection pending.	
	T_INCON	Incoming connection pending.	
	T_DATAXFER	Data transfer.	
	T_OUTREL	Outgoing direction orderly release sent.	
	T_INREL	Incoming direction orderly release received.	
	If the provider is undergoing a state transition when t_getstate() is called, the function will fail.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	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).	
	TSTATECHNG	The transport provider is undergoing a transient state change.	
	TSYSERR	A system error has occurred during execution of this function.	
TLI COMPATIBILITY	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.		
Interface Header	The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:		
	<pre>#include <tiuser.h></tiuser.h></pre>		

614 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

#### t\_getstate(3NSL)

# Error Description Values The t\_errno value that can be set by the XTI interface and cannot be set by the TLI interface is: TPROTO TPROTO

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

SEE ALSO

t\_open(3NSL), attributes(5)

## t\_listen(3NSL)

NAME	t_listen – listen for a connection indication		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_listen(int fd, struct t_call *call);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	This function listens for a connection indication from a calling transport user. The argument <i>fd</i> identifies the local transport endpoint where connection indications arrive, and on return, <i>call</i> contains information describing the connection indication. The parameter <i>call</i> points to a t_call structure which contains the following members:		
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; In call, addr returns the protocol address of the calling transport user. This address is in a format usable in future calls to t_connect(3NSL). Note, however that t_connect(3NSL) may fail for other reasons, for example TADDRBUSY. opt returns options associated with the connection indication, udata returns any user data sent by the caller on the connection request, and sequence is a number that uniquely identifies the returned connection indication. The value of sequence enables the user to listen for multiple connection indications before responding to any of them.</pre>		
	Since this function returns values for the <i>addr</i> , <i>opt</i> and <i>udata</i> fields of <i>call</i> , the <i>maxlen</i> field of each must be set before issuing the t_listen() to indicate the maximum size of the buffer for each. If the <i>maxlen</i> field of <i>call</i> $\rightarrow$ <i>addr</i> , <i>call</i> $\rightarrow$ <i>opt</i> or <i>call</i> $\rightarrow$ <i>udata</i> is set to zero, no information is returned for this parameter.		
	By default, t_listen() executes in synchronous mode and waits for a connection indication to arrive before returning to the user. However, if O_NONBLOCK is set via t_open(3NSL) or fcntl(2), t_listen() executes asynchronously, reducing to a poll for existing connection indications. If none are available, it returns -1 and sets t_errno to TNODATA.		
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE, T_INCON		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF T	he specified file descriptor does not refer to a transport endpoint.	
	TBADQLEN T	he argument <i>qlen</i> of the endpoint referenced by <i>fd</i> is zero.	
'			

	TBUFOVFLW	The number of bytes allocated 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, as seen by the user, changes to T_INCON, and the connection indication information to be returned in <i>call</i> is discarded. The value of <i>sequence</i> returned can be used to do a t_snddis(3NSL).
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
	TNODATA	O_NONBLOCK was set, but no connection indications had been queued.
	TNOTSUPPORT	This function is not supported by the underlying transport provider.
	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).
	TQFULL	The maximum number of outstanding connection indications has been reached for the endpoint referenced by $fd$ . Note that a subsequent call to t_listen() may block until another incoming connection indication is available. This can only occur if at least one of the outstanding connection indications becomes no longer outstanding, for example through a call to t_accept(3NSL).
	TSYSERR	A system error has occurred during execution of this function.
TLI COMPATIBILITY	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.	
Interface Header	The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:	
	<pre>#include <tiuser.h></tiuser.h></pre>	
Error Description Values	The t_errno values TPROTO, TBADQLEN, and TQFULL can be set by the XTI interface but not 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.	
Option Buffers		options in an opt buffer is dictated by the transport provider. Unlike the TLI interface does not fix the buffer format.

## t\_listen(3NSL)

 $\label{eq:attributes} \textbf{ATTRIBUTES} ~|~ \textbf{See attributes}(5) ~ for descriptions of the following attributes:$ 

	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT Level	Safe
		·
SEE ALSO	<pre>fcntl(2), t_accept(3NSL), t_alloc(3NS t_open(3NSL), t_optmgmt(3NSL), t_rcv attributes(5)</pre>	
WARNINGS	Some transport providers do not differentia connection itself. If this is the case, a succes existing connection.	

NAME	t_look – look at the current event on a transport endpoint		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_look(int fd);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	This function returns the current event on the transport endpoint specified by <i>fd</i> . This function enables a transport provider to notify a transport user of an asynchronous event when the user is calling functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, TLOOK, on the current or next function to be executed.		
	This function also enables a transference asynchronous events.	ansport user to poll a transport endpoint periodically for	
RETURN VALUES	Upon success, t_look() returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists. One of the following events is returned:		
	T_LISTEN	Connection indication received.	
	T_CONNECT	Connect confirmation received.	
	T_DATA	Normal data received.	
	T_EXDATA	Expedited data received.	
	T_DISCONNECT	Disconnection received.	
	T_UDERR	Datagram error indication.	
	T_ORDREL	Orderly release indication.	
	T_GODATA	Flow control restrictions on normal data flow that led to a TFLOW error have been lifted. Normal data may be sent again.	
	T_GOEXDATA	Flow control restrictions on expedited data flow that led to a TFLOW error have been lifted. Expedited data may be sent again.	
	On failure, -1 is returned and t_errno is set to indicate the error.		
VALID STATES	ALL - apart from T_UNINIT.		
ERRORS	On failure, t_errno is set to	one of the following:	

t\_look(3NSL)

	TBADF	The specifie transport en	ed file descriptor does not refer to a ndpoint.
	TPROTO	This error in been detecte	ndicates that a communication problem has ted between XTI and the transport provider here is no other suitable XTI error
	TSYSERR	A system err function.	rror has occurred during execution of this
TLI COMPATIBILITY	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.		
Interface Header	The XTI interfaces use the hea header. They should use the h		. h. TLI interfaces should <i>not</i> use this
	<pre>#include <tiuser.h></tiuser.h></pre>		
Return Values	The return values that are defined by the XTI interface and cannot be returned by the TLI interface are:		
	T_GODATA T_GOEXDATA		
Error Description Values	The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:		
	TPROTO		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		ATTRIBUTE VALUE
	MT Level		Safe
SEE ALSO	t_open(3NSL), t_snd(3NSL), t_sndudata(3NSL), attributes(5)		

NAME	t_open – establish a transport endpoint		
SYNOPSIS	<pre>#include <xti.h> #include <fcntl.h></fcntl.h></xti.h></pre>		
	<pre>int t_open(const char *name, int oflag, struct t_info *info);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	The t_open() function must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by supplying a transport provider identifier that indicates a particular transport provider, that is, transport protocol, and returning a file descriptor that identifies that endpoint.		
	The argument <i>name</i> points to a transport provider identifier and <i>oflag</i> identifies any open flags, as in open(2). The argument <i>oflag</i> is constructed from O_RDWR optionally bitwise inclusive-OR'ed with O_NONBLOCK. These flags are defined by the header <fcntl.h>. The file descriptor returned by t_open() will be used by all subsequent functions to identify the particular local transport endpoint.</fcntl.h>		
	This function also returns various default characteristics of the underlying transport protocol by setting fields in the t_info structure. This argument points to a t_info which contains the following members:		
	<pre>t_scalar_t addr;  /* max size of the transport protocol address */ t_scalar_t options;  /* max number of bytes of */</pre>		
	t_scalar_t tsdu;		
	t_scalar_t etsdu;		
	t_scalar_t connect		
	t_scalar_t discon;		
	t_scalar_t servtyp		
	t_scalar_t flags;		
	The values of the fields have the following meanings:addrA value greater than zero (T_NULL) indicates the maximum size a transport protocol address and a value of -2 (T_INVALID) specifies that the transport provider does not provide user access to transport protocol addresses.optionsA value greater than zero (T_NULL) indicates the maximum number of bytes of protocol-specific options supported by the		

		provider, and a value of -2 (T_INVALID) specifies that the transport provider does not support user-settable options.
	tsdu	A value greater than zero (T_NULL specifies the maximum size of a transport service data unit (TSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 (T_INFINITE) specifies that there is no limit to the size of a TSDU; and a value of -2 (T_INVALID) specifies that the transfer of normal data is not supported by the transport provider.
	etsdu	A value greater than zero (T_NULL) specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 (T_INFINITE) specifies that there is no limit on the size of an ETSDU; and a value of -2 (T_INVALID) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers.
	connect	A value greater than zero (T_NULL) specifies the maximum amount of data that may be associated with connection establishment functions, and a value of -2 (T_INVALID) specifies that the transport provider does not allow data to be sent with connection establishment functions.
l	discon	If the T_ORDRELDATA bit in flags is clear, a value greater than zero (T_NULL) specifies the maximum amount of data that may be associated with the t_snddis(3NSL) and t_rcvdis(3NSL) functions, and a value of $-2$ (T_INVALID) specifies that the transport provider does not allow data to be sent with the abortive release functions. If the T_ORDRELDATA bit is set in flags, a value greater than zero (T_NULL) specifies the maximum number of octets that may be associated with the t_sndreldata(), t_rcvreldata(), t_snddis(3NSL) and t_rcvdis(3NSL) functions.
5	servtype	This field specifies the service type supported by the transport provider, as described below.
)	flags	This is a bit field used to specify other information about the communications provider. If the T_ORDRELDATA bit is set, the communications provider supports user data to be sent with an orderly release. If the T_SENDZERO bit is set in flags, this indicates the underlying transport provider supports the sending of zero-length TSDUS.
1		

	If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the t_alloc(3NSL) function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.	
	The <i>servtype</i> field of <i>info</i> specifies one of the following values on return:	
	T_COTSThe transport provider supports a connection-mode service but does not support the optional orderly release facility.	
	T_COTS_ORD	The transport provider supports a connection-mode service with the optional orderly release facility.
	T_CLTS	The transport provider supports a connectionless-mode service. For this service type, t_open() will return -2 (T_INVALID) for <i>etsdu</i> , <i>connect</i> and <i>discon</i> .
	A single transport	endpoint may support only one of the above services at one time.
	If <i>info</i> is set to a nuby t_open().	Ill pointer by the transport user, no protocol information is returned
<b>RETURN VALUES</b>	A valid file descriptor is returned upon successful completion. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.	
VALID STATES	T_UNINIT.	
ERRORS	On failure, t_errno is set to the following:	
	TBADFLAG	An invalid flag is specified.
	TBADNAME	Invalid transport provider name.
	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.
TLI COMPATIBILITY	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.	
Interface Header	The XTI interfaces use the xti.h TLI interfaces should <i>not</i> use this header. They should use the header:	
	#include <tiuser.h< th=""><th>&gt;</th></tiuser.h<>	>
Error Description Values		ies TPROTO and TBADNAME can be set by the XTI interface but

t\_open(3NSL)

**Notes** | For TLI , the t\_info structure referenced by *info* lacks the following structure member:

t\_scalar\_t flags; /\* other info about the transport provider \*/

This member was added to struct t\_info in the XTI interfaces.

When a value of -1 is observed as the return value in various t\_info structure members, it signifies that the transport provider can handle an infinite length buffer for a corresponding attribute, such as address data, option data, TSDU (octet size), ETSDU (octet size), connection data, and disconnection data. The corresponding structure members are addr, options, tsdu, estdu, connect, and discon, respectively.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE	
MT Level	Safe	

**SEE ALSO** open(2), attributes(5)

NAME	t_optmgmt – manage options for a transport endpoint		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_optmgmt(int fd, const struct t_optmgmt *req, struct t_optmgmt</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	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.		
	The <i>req</i> and <i>ret</i> arguments point to a t_optmgmt structure containing the following members:		
	<pre>struct netbuf opt; t_scalar_t flags; The opt field identifies protocol options and the flags field is used to specify the action to take with those options.</pre>		
	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.		
	Each option in the options buffer is of the form struct t_opthdr possibly followed by an option value.		
	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.		
	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.		
	Networking Library Functions 625		

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(tohp)	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 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_optmgmt() 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\_optmgmt() with the T\_CURRENT flag set.

The *flags* field of *req* must specify one of the following actions:

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* $\rightarrow$ *opt.buf* and *req* $\rightarrow$ *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* $\rightarrow$ *flags*.

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

T\_NEGOTIATE

		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 <i>req</i> 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 <i>status</i> 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 <i>status</i> 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 <i>ret</i> $\rightarrow$ <i>flags</i> . 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.
ľ	DEFAULT	This action enables the transport user to retrieve the default option values. The user specifies the options of interest in $req \rightarrow opt.buf$ . 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 $ret \rightarrow opt.buf$ .
		The <i>status</i> 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  $ret \rightarrow flags$ . 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,  $ret \rightarrow opt.maxlen$  must be given at least the value *info\rightarrow options* before the call. See t getinfo(3NSL) and t open(3NSL).

This action enables the transport user to retrieve the currently effective option values. The user specifies the options of interest in *req* $\rightarrow$ *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* $\rightarrow$ *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\_optmgmt() 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\_optmgmt() call only options of one level may be addressed, this

T CURRENT

L_opunginu(SNSL)			
			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_optmgmt() 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	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set to indicate an error.		
VALID STATES	ALL - apart from T_UNINIT.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file des	criptor does not refer to a transport endpoint.
	TBADFLAG	An invalid flag was s	pecified.
	TBADOPT	The specified options illegal information.	were in an incorrect format or contained
	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 sup	ported by the transport provider.

	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).
	TSYSERR	A system error has occurred during execution of this function.
TLI COMPATIBILITY	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.	
Interface Header	The XTI interfaces header. They shou	use the header file, xti.h. TLI interfaces should <i>not</i> use this ld use the header:
	#include <tiuser.]< th=""><th>n&gt;</th></tiuser.]<>	n>
Error Description Values	The t_errno value TPROTO can be set by the XTI interface but not by the TLI interface.	
	The t_errno 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 maxlen field of the corresponding buffer has been set to zero.
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. The macros T_OPT_DATA, T_OPT_NEXTHDR, and T_OPT_FIRSTHDR described for XTI are not available for use by TLI interfaces.	
Actions	The semantic meaning of various action values for the flags 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 evaluate the requested options and negotiate the values, returning the negotiated values through <i>ret</i> .
	T_CHECK	This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the flags field of <i>ret</i> will have either T_SUCCESS or T_FAILURE set to indicate to the user whether the options are supported. These flags are only meaningful for the T_CHECK request.

T\_DEFAULTThis action enables a user to retrieve the default options supported<br/>by the transport provider into the opt field of ret. In req, the len<br/>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.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

SEE ALSO close(2), poll(2), select(3C), t\_accept(3NSL), t\_alloc(3NSL), t\_bind(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)

NAME	t_rcv – receive data or expedited data sent over a connection		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_rcv(int fd, void *buf, unsigned int nbytes, int *flags);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	This function receives either normal or expedited data. The argument $fd$ identifies the local transport endpoint through which data will arrive, <i>buf</i> points to a receive buffer where user data will be placed, and <i>nbytes</i> specifies the size of the receive buffer. The argument <i>flags</i> may be set on return from t_rcv() and specifies optional flags as described below.		
	By default, t_rcv() operates in synchronous mode and will wait for data to arrive if none is currently available. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_rcv() will execute in asynchronous mode and will fail if no data is available. See TNODATA below.		
	On return from the call, if T_MORE is set in <i>flags</i> , this indicates that there is more data, and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple t_rcv() calls. In the asynchronous mode, or under unusual conditions (for example, the arrival of a signal or T_EXDATA event), the T_MORE flag may be set on return from the t_rcv() call even when the number of bytes received is less than the size of the receive buffer specified. Each t_rcv() with the T_MORE flag set indicates that another t_rcv() must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a t_rcv() call with the T_MORE flag not set. 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 should be ignored. If <i>nbytes</i> is greater than zero on the call to t_rcv(), t_rcv() will return 0 only if the end of a TSDU is being returned to the user.		
	On return, the data is expedited if T_EXPEDITED is set in flags. If T_MORE is also set, it indicates that the number of expedited bytes exceeded nbytes, a signal has interrupted the call, or that an entire ETSDU was not available (only for transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will be returned by subsequent calls to t_rcv() which will return with T_EXPEDITED set in flags. The end of the ETSDU is identified by the return of a t_rcv() call with T_EXPEDITED set and T_MORE cleared. If the entire ETSDU is not available it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.		

t_rcv(3NSL)				
	If a signal arrives, t_rcv() returns, giving the user any data currently available. If no data is available, t_rcv() returns -1, sets t_errno to TSYSERR and errno to EINTR. If some data is available, t_rcv() returns the number of bytes received and T_MORE is set in flags.			
	In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the T_DATA or T_EXDATA events using the t_look(3NSL) function. Additionally, the process can arrange to be notified by means of the EM interface.			
RETURN VALUES		npletion, $t_rcv()$ returns the number of bytes received. Otherwise, ure and $t_errno$ is set to indicate the error.		
VALID STATES	T_DATAXFER, T_C	DUTREL.		
ERRORS	On failure, t_err	mo is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.		
	TLOOK An asynchronous event has occurred on this transport endpoint and requires immediate attention.			
	TNODATA O_NONBLOCK was set, but no data is currently available from the transport provider.			
	TNOTSUPPORT This function is not supported by the underlying transport provider.			
	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.		
TLI COMPATIBILITY	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.			
Interface Header	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< th=""><th>&gt;</th></tiuser.h<>	>		
Error Description Values	The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:			
	TPROTO			
ATTRIBUTES	See attributes(	(5) for descriptions of the following attributes:		

634 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

t\_rcv(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

SEE ALSO fcntl(2), t\_getinfo(3NSL), t\_look(3NSL), t\_open(3NSL), t\_snd(3NSL), attributes(5)

t	rcvconnect	(3NSL)

NAME	t_rcvconnect – receive the confirmation from a connection request
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_rcvconnect(int fd, struct t_call *call);</pre>
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.
	This function enables a calling transport user to determine the status of a previously sent connection request and is used in conjunction with t_connect(3NSL) to establish a connection in asynchronous mode, and to complete a synchronous t_connect(3NSL) call that was interrupted by a signal. The connection will be established on successful completion of this function.
	The argument <i>fd</i> identifies the local transport endpoint where communication will be established, and <i>call</i> contains information associated with the newly established connection. The argument <i>call</i> points to a t_call structure which contains the following members:
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; In call, addr returns the protocol address associated with the responding transport endpoint, opt presents any options associated with the connection, udata points to optional user data that may be returned by the destination transport user during connection establishment, and sequence has no meaning for this function.</pre>
	The <i>maxlen</i> field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, <i>maxlen</i> can be set to zero, in which case no information to this specific argument is given to the user on the return from t_rcvconnect(). If <i>call</i> is set to NULL, no information at all is returned. By default, t_rcvconnect() executes in synchronous mode and waits for the connection to be established before returning. On return, the <i>addr</i> , <i>opt</i> and <i>udata</i> fields reflect values associated with the connection.
	If O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_rcvconnect() executes in asynchronous mode, and reduces to a poll for existing connection confirmations. If none are available, t_rcvconnect() fails and returns immediately without waiting for the connection to be established. See TNODATA below. In this case, t_rcvconnect() must be called again to complete the connection establishment phase and retrieve the information returned in <i>call</i> .
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set to indicate an error.
VALID STATES	T_OUTCON.

t\_rcvconnect(3NSL)

ERRORS On failure, t errno is set to one of the following: TBADF The specified file descriptor does not refer to a transport endpoint. TBUFOVFLW The number of bytes allocated for an incoming argument (*maxlen*) is greater than 0 but not sufficient to store the value of that argument, and the connection information to be returned in *call* will be discarded. The provider's state, as seen by the user, will be changed to T DATAXFER. TLOOK An asynchronous event has occurred on this transport connection and requires immediate attention. TNODATA O NONBLOCK was set, but a connection confirmation has not yet arrived. This function is not supported by the underlying transport TNOTSUPPORT provider. TOUTSTATE The communications endpoint referenced by *fd* 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 (t errno). TSYSERR A system error has occurred during execution of this function. TLI The XTI and TLI interface definitions have common names but use different header COMPATIBILITY files. This, and other semantic differences between the two interfaces are described in the subsections below. **Interface Header** The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header: #include<tiuser.h> **Error Description** The t errno value TPROTO can be set by the XTI interface but not by the TLI Values interface. At 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. **ATTRIBUTES** See attributes(5) for descriptions of the following attributes: ATTRIBUTE TYPE ATTRIBUTE VALUE MT Level Safe

Networking Library Functions 637

t_rcvconnect(3NSL) SEE ALSO	<pre>fcntl(2), t_accept(3NSL), t_alloc(3NSL), t_bind(3NSL), t_connect(3NSL), t_listen(3NSL), t_open(3NSL), t_optmgmt(3NSL), attributes(5)</pre>
	c_fisten(SN3E), c_open(SN3E), c_opengine(SN3E), accribites(3)

NAME	t_rcvdis – retrieve	information from disconnection	
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	int <b>t_rcvdis</b> (i	<pre>.nt fd, struct t_discon *discon);</pre>	
DESCRIPTION	This routine is part of the XTI interfaces which evolved from the TLI interfaces. The represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name a XTI routine, the tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	data sent with the	ted to identify the cause of a disconnection and to retrieve any user disconnection. The argument <i>fd</i> identifies the local transport the connection existed, and <i>discon</i> points to a t_discon structure owing members:	
	struct netbuf uda	ta;	
	<pre>int reason; int sequence; The field reason specifies the reason for the disconnection through a protocol-dependent reason code, udata identifies any user data that was sent with the disconnection, and sequence may identify an outstanding connection indication with which the disconnection is associated. The field sequence is only meaningful when t_rcvdis() is issued by a passive transport user who has executed one or more t_listen(3NSL) functions and is processing the resulting connection indications. If a disconnection indication occurs, sequence can be used to identify which of the outstanding connection indications is associated with the disconnection.</pre>		
		ection indication by means of t_listen(3NSL), and <i>discon</i> is a null vill be unable to identify with which connection indication the ssociated.	
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_DATAXFER, T_OUTCON, T_OUTREL, T_INREL, T_INCON(ocnt > 0).		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TBUFOVFLW	The number of bytes allocated for incoming data ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the data. If <i>fd</i> is a passive endpoint with <i>ocnt</i> > 1, it remains in state T_INCON; otherwise, the endpoint state is set to T_IDLE.	

t\_rcvdis(3NSL)

	TNODIS	No disconnection indic transport endpoint.	ation currently exists on the specified
	TNOTSUPPORT	This function is not sup provider.	pported by the underlying transport
	TOUTSTATE	The communications en states in which a call to	ndpoint referenced by $fd$ is not in one of the this function is valid.
	TPROTO		t a communication problem has been and the transport provider for which there is rror (t_errno).
	TSYSERR	A system error has occu	urred during execution of this function.
TLI COMPATIBILITY		er semantic differences b	common names but use different header between the two interfaces are described in
Interface Header	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></tiuser.h>	>	
Error Description			ATE can be set by the XTI interface but not
Values	The t_errno values TPROTO and TOUTSTATE can be set by the XTI interface but not by the TLI interface.		
	circumstances than		this routine can set under different BUFOVFLW. It can be returned even when fer has been set to zero.
ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATTF	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe
SEE ALSO	t_alloc(3NSL),t t_snddis(3NSL),		isten(3NSL), t_open(3NSL),

NAME	t_rcvrel – acknowledge receipt of an orderly release indication		
SYNOPSIS	#include <xti.h></xti.h>		
	int <b>t_rcvrel</b> (	int fd);	
DESCRIPTION	represents the future supported for com XTI routine, the t	rt of the XTI interfaces which evolved from the TLI interfaces. XTI ure evolution of these interfaces. However, TLI interfaces are npatibility. When using a TLI routine that has the same name as an iuser.h header file must be used. Refer to the TLI section for a description of differences between the two interfaces.	
	of data transfer. The connection exists. more data by mean terror set to TOUT connection if terror set to TOUT connection if terror or provider returned	Seed to receive an orderly release indication for the incoming direction the argument $fd$ identifies the local transport endpoint where the After receipt of this indication, the user may not attempt to receive ns of t_rcv(3NSL) or t_rcvv(). Such an attempt will fail with CSTATE. However, the user may continue to send data over the ndre1(3NSL) has not been called by the user. This function is an f the transport provider, and is only supported if the transport service type T_COTS_ORD on t_open(3NSL) or t_getinfo(3NSL). t may be associated with the orderly release indication is discarded () is called.	
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set to indicate an error.		
VALID STATES	T_DATAXFER, T_OUTREL.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.	
	TNOREL	No orderly release indication currently exists on the specified transport endpoint.	
	TNOTSUPPORT	This function is not supported by the underlying 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 (t_errno).	
	TSYSERR	A system error has occurred during execution of this function.	
TLI COMPATIBILITY		nterface definitions have common names but use different header ner semantic differences between the two interfaces are described in clow.	

## t\_rcvrel(3NSL)

Interface Header	The XTI interfaces use the header file, xti. header. They should use the header:	h. TLI interfaces should <i>not</i> use this
	#include <tiuser.h></tiuser.h>	
Error Description Values	The t_errno values that can be set by the interface are:	XTI interface and cannot be set by the TLI
	TPROTO TOUTSTATE	
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT Level	Safe
SEE ALSO	t_getinfo(3NSL), t_open(3NSL), t_snd	drel(3NSL), attributes(5)

		t_rcvreldata(3NSL)	
NAME	t_rcvreldata – receive an orderly release indication or confirmation containing user data		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	int <b>t_rcvreld</b> a	<pre>ata(int fd, struct t_discon *discon);</pre>	
DESCRIPTION	of data transfer an identifies the local	sed to receive an orderly release indication for the incoming direction ad to retrieve any user data sent with the release. The argument <i>fd</i> transport endpoint where the connection exists, and <i>discon</i> points to ture containing the following members:	
	data by means of <i>t_error</i> set to TOUT	ta; r receipt of this indication, the user may not attempt to receive more t_rcv(3NSL) or t_rcvv(3NSL) Such an attempt will fail with 'STATE. However, the user may continue to send data over the ndrel(3NSL) or t_sndreldata (3N) has not been called by the	
	The field <i>reason</i> specifies the reason for the disconnection through a protocol-dependent <i>reason code</i> , and <i>udata</i> identifies any user data that was sent with the disconnection; the field <i>sequence</i> is not used.		
	If a user does not care if there is incoming data and does not need to know the value of <i>reason</i> , <i>discon</i> may be a null pointer, and any user data associated with the disconnection will be discarded.		
	If $discon \rightarrow udata.maxlen$ is greater than zero and less than the length of the value, t_rcvreldata() fails with t_errno set to TBUFOVFLW.		
	providers of servic field returned by t supports orderly r	a optional service of the transport provider, only supported by ce type T_COTS_ORD. The flag T_ORDRELDATA in the <i>info</i> $\rightarrow$ <i>flag</i> t_open(3NSL) or t_getinfo(3NSL) indicates that the provider release user data; when the flag is not set, this function behaves like and no user data is returned.	
	This function may	not be available on all systems.	
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_DATAXFER, T_C	DUTREL.	
ERRORS	On failure, t_err	no is set to one of the following:	
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TBUFOVFLW	The number of bytes allocated for incoming data (maxlen) is greater than 0 but not sufficient to store the data, and the disconnection information to be returned in <i>discon</i> will be	

t\_rcvreldata(3NSL)

- ,			
		discarded. The provide as if the data was succe	r state, as seen by the user, will be changed ssfully retrieved.
	TLOOK	An asynchronous even and requires immediate	t has occurred on this transport endpoint e attention.
	TNOREL	No orderly release indi transport endpoint.	cation currently exists on the specified
	TNOTSUPPORT	Orderly release is not s provider.	upported by the underlying transport
	TOUTSTATE	The communications er states in which a call to	ndpoint referenced by <i>fd</i> is not in one of the this function is valid.
	TPROTO		t a communication problem has been and the transport provider for which there is cror (t_errno).
	TSYSERR	A system error has occu	urred during execution of this function.
TLI COMPATIBILITY		-	art of this routine was defined.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	AII	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		ATTRIBUTE VALUE Safe
SEE ALSO	MT Level t_getinfo(3NSI		
SEE ALSO NOTES	MT Level t_getinfo(3NSI t_sndrel(3NSL) The interfaces t_ specific transport	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.
	MT Level t_getinfo(3NSL t_sndrel(3NSL) The interfaces t_ specific transport These interfaces a	L), t_open(3NSL), t_sno ), attributes(5) sndreldata(3NSL) and called "minimal OSI," wł	Safe dreldata(3NSL), t_rcvrel(3NSL), t_rcvreldata() are only for use with a nich is not available on the Solaris platform.

NAME	t_rcvudata – receive a data unit		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_rcvudata(int fd, struct t_unitdata *unitdata, int *flags);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	This function is used in connectionless-mode to receive a data unit from another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be received, <i>unitdata</i> holds information associated with the received data unit, and <i>flags</i> is set on return to indicate that the complete data unit was not received. The argument <i>unitdata</i> points to a t_unitdata structure containing the following members:		
	struct netbuf addr; struct netbuf opt; struct netbuf udata;		
	The <i>maxlen</i> field of <i>addr</i> , <i>opt</i> and <i>udata</i> must be set before calling this function to indicate the maximum size of the buffer for each. If the <i>maxlen</i> field of <i>addr</i> or <i>opt</i> is set to zero, no information is returned in the <i>buf</i> field of this parameter.		
	On return from this call, <i>addr</i> specifies the protocol address of the sending user, <i>opt</i> identifies options that were associated with this data unit, and <i>udata</i> specifies the user data that was received.		
	By default, t_rcvudata() operates in synchronous mode and will wait for a data unit to arrive if none is currently available. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_rcvudata() will execute in asynchronous mode and will fail if no data units are available.		
	If the buffer defined in the <i>udata</i> field of <i>unitdata</i> is not large enough to hold the current data unit, the buffer will be filled and T_MORE will be set in <i>flags</i> on return to indicate that another t_rcvudata() should be called to retrieve the rest of the data unit. Subsequent calls to t_rcvudata() will return zero for the length of the address and options until the full data unit has been received.		
	If the call is interrupted, t_rcvudata() will return EINTR and no datagrams will have been removed from the endpoint.		
<b>RETURN VALUES</b>	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE.		

t\_rcvudata(3NSL)

ERRORS	On failure, t_errno is set to one of the following:	
	TBADF	The specified file descriptor does not refer to a transport endpoint.
	TBUFOVFLW	The number of bytes allocated for the incoming protocol address or options ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the information. The unit data information to be returned in <i>unitdata</i> will be discarded.
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
	TNODATA	O_NONBLOCK was set, but no data units are currently available from the transport provider.
	TNOTSUPPORT	This function is not supported by the underlying transport provider.
	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).
	TSYSERR	A system error has occurred during execution of this function.
TLI COMPATIBILITY	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.	
Interface Header	The XTI interfaces header. They shou	use the header file, xti.h. TLI interfaces should <i>not</i> use this ld use the header:
	#include <tiuser.h></tiuser.h>	>
Error Description Values	The t_errno valu interface are:	ues that can be set by the XTI interface and cannot be set by the TLI
	TPROTO TOUTSTATE	
	XTI counterpart is	that this routine can return under different circumstances than its TBUFOVFLW. It can be returned even when the maxlen field of the ffer has been set to zero.
<b>Option Buffers</b>	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.	
ATTRIBUTES	See attributes(	5) for descriptions of the following attributes:

t\_rcvudata(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

### SEE ALSO fcntl(2), t\_alloc(3NSL), t\_open(3NSL), t\_rcvuderr(3NSL), t\_sndudata(3NSL), attributes(5)

# t\_rcvuderr(3NSL)

NAME	t_rcvuderr – receive a unit data error indication		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	int <b>t_rcvuderr</b>	(int fd, struct t_uderr *uderr);	
DESCRIPTION	This routine is part of the XTI interfaces which evolved from the TLI interfaces. XT 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 a XTI routine, the tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interface		
	This function is used in connectionless-mode to receive information concerning an error on a previously sent data unit, and should only be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. The argument $fd$ identifies the local transport endpoint through which the error report will be received, and <i>uderr</i> points to a t_uderr structure containing the following members:		
	<pre>struct netbuf addr; struct netbuf opt; t_scalar_t error; The maxlen field of addr and opt must be set before calling this function to indicate the maximum size of the buffer for each. If this field is set to zero for addr or opt, no information is returned in the buf field of this parameter.</pre>		
	On return from this call, the <i>addr</i> structure specifies the destination protocol address of the erroneous data unit, the <i>opt</i> structure identifies options that were associated with the data unit, and error specifies a protocol-dependent error code.		
	If the user does not care to identify the data unit that produced an error, <i>uderr</i> may be set to a null pointer, and t_rcvuderr() will simply clear the error indication without reporting any information to the user.		
RETURN VALUES		ompletion, a value of 0 is returned. Otherwise, a value of –1 is rrno is set to indicate an error.	
VALID STATES	T_IDLE.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TBUFOVFLW	The number of bytes allocated for the incoming protocol address or options ( <i>maxlen</i> ) is greater than 0 but not sufficient to store the information. The unit data error information to be returned in <i>uderr</i> will be discarded.	
	TNOTSUPPORT	This function is not supported by the underlying transport provider.	

TNOUDERR         No unit data error indication currently exists on the specified transport endpoint.		
TOUTSTATEThe communications endpoint referenced by $fd$ is not in one of the states in which a call to this function is valid.		
TPROTO	detected between XTI a	t a communication problem has been nd the transport provider for which there is ror (t_errno).
TSYSERR	A system error has occu	urred during execution of this function.
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.		
The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:		
#include <tius< th=""><th>er h</th><th></th></tius<>	er h	
The t_errno values TPROTO and TOUTSTATE can be set by the XTI interface but not by the TLI interface.		
A + errno value	that this routine can retu	we are dealed ifferent since an element of the second
XTI counterpart is		rn under different circumstances than its eturned even when the maxlen field of the
XTI counterpart is corresponding buff The format of the o	TBUFOVFLW. It can be r fer has been set to zero.	eturned even when the maxlen field of the is dictated by the transport provider. Unlike
XTI counterpart is corresponding buff The format of the o the XTI interface, t	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format.
XTI counterpart is corresponding buff The format of the of the XTI interface, t See attributes(	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes:
XTI counterpart is corresponding buff The format of the of the XTI interface, t See attributes(	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format.
XTI counterpart is corresponding buff The format of the of the XTI interface, t See attributes(	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes:
XTI counterpart is corresponding buff The format of the o the XTI interface, t See attributes( MT Level	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes: ATTRIBUTE VALUE Safe
XTI counterpart is corresponding buff The format of the o the XTI interface, t See attributes( MT Level	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the <b>RIBUTE TYPE</b>	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes: ATTRIBUTE VALUE Safe
XTI counterpart is corresponding buff The format of the o the XTI interface, t See attributes( MT Level	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the <b>RIBUTE TYPE</b>	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes: ATTRIBUTE VALUE Safe
XTI counterpart is corresponding buff The format of the o the XTI interface, t See attributes( MT Level	TBUFOVFLW. It can be r fer has been set to zero. options in an opt buffer he TLI interface does not 5) for descriptions of the <b>RIBUTE TYPE</b>	eturned even when the maxlen field of the is dictated by the transport provider. Unlike t fix the buffer format. following attributes: ATTRIBUTE VALUE Safe
	TOUTSTATE TPROTO TSYSERR The XTI and TLI ir files. This, and oth the subsections bel The XTI interfaces header. They shou #include <tius The t_errno value by the TLI interface</tius 	transport endpoint.TOUTSTATEThe communications er states in which a call toTPROTOThis error indicates that detected between XTI a no other suitable XTI erTSYSERRA system error has occuThe XTI and TLI interface definitions have files. This, and other semantic differences b the subsections below.The XTI interfaces use the header file, xti header. They should use the header:#include <tiuser.h>The t_errno values TPROTO and TOUTST. by the TLI interface.</tiuser.h>

## t\_rcvv(3NSL)

NAME	t_rcvv – receive data or expedited data sent over a connection and put the data into one or more non-contiguous buffers
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_rcvv(int fd, struct t_iovec *iov, unsigned int iovcount, int  *flags);</pre>
DESCRIPTION	This function receives either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint through which data will arrive, <i>iov</i> points to an array of buffer address/buffer size pairs ( <i>iov_base, iov_len</i> ). The t_rcvv() function receives data into the buffers specified by <i>iov0.iov_base, iov1.iov_base</i> , through <i>iov</i> [ <i>iovcount-1</i> ]. <i>iov_base</i> , always filling one buffer before proceeding to the next.
	Note that the limit on the total number of bytes available in all buffers passed:
	<i>iov</i> (0). <i>iov_len</i> + + <i>iov</i> ( <i>iovcount-1</i> ). <i>iov_len</i> )may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.
	The argument iovcount contains the number of buffers which is limited to T_IOV_MAX, which is an implementation-defined value of at least 16. If the limit is exceeded, the function will fail with TBADDATA.
	The argument flags may be set on return from $t_rcvv()$ and specifies optional flags as described below.
	By default, t_rcvv() operates in synchronous mode and will wait for data to arrive if none is currently available. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_rcvv() will execute in asynchronous mode and will fail if no data is available. See TNODATA below.
	On return from the call, if T_MORE is set in flags, this indicates that there is more data, and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple t_rcvv() or t_rcv(3NSL) calls. In the asynchronous mode, or under unusual conditions (for example, the arrival of a signal or T_EXDATA event), the T_MORE flag may be set on return from the t_rcvv() call even when the number of bytes received is less than the total size of all the receive buffers. Each t_rcvv() with the T_MORE flag set indicates that another t_rcvv() must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a t_rcvv() call with the T_MORE flag not set. 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 should be ignored. If the amount of buffer space passed in <i>iov</i> is greater than zero on the call to t_rcvv(), then t_rcvv() will return 0 only if the end of a TSDU is being returned to the user.

	On return, the data is expedited if T_EXPEDITED is set in flags. If T_MORE is also set, it indicates that the number of expedited bytes exceeded nbytes, a signal has interrupted the call, or that an entire ETSDU was not available (only for transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will be returned by subsequent calls to t_rcvv() which will return with T_EXPEDITED set in flags. The end of the ETSDU is identified by the return of a t_rcvv() call with T_EXPEDITED set and T_MORE cleared. If the entire ETSDU is not available it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.		
	If a signal arrives, t_rcvv() returns, giving the user any data currently available. If no data is available, t_rcvv() returns -1, sets t_errno to TSYSERR and errno to EINTR. If some data is available, t_rcvv() returns the number of bytes received and T_MORE is set in flags.		
	In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the T_DATA or T_EXDATA events using the t_look(3NSL) function. Additionally, the process can arrange to be notified via the EM interface.		
RETURN VALUES	On successful completion, t_rcvv() returns the number of bytes received. Otherwise, it returns -1 on failure and t_errno is set to indicate the error.		
VALID STATES	T_DATAXFER, T_0	OUTREL.	
ERRORS	On failure, t_errno is set to one of the following:		
	TBADDATA	<i>iovcount</i> is greater than T_IOV_MAX.	
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.	
	TNODATA	O_NONBLOCK was set, but no data is currently available from the transport provider.	
	TNOTSUPPORT	This function is not supported by the underlying transport provider.	
	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).	
	TSYSERR	A system error has occurred during execution of this function.	
TLI	In the TLI interfac	e definition, no counterpart of this routine was defined.	
COMPATIBILITY ATTRIBUTES	See attributes	(5) for descriptions of the following attributes:	

#### t\_rcvv(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

#### SEE ALSO fcntl(2), t\_getinfo(3NSL), t\_look(3NSL), t\_open(3NSL), t\_rcv(3NSL), t\_snd(3NSL), t\_sndv(3NSL), attributes(5)

```
t_rcvvudata(3NSL)
```

NAME	t_rcvvudata – receive a data unit into one or more noncontiguous buffers
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_rcvvudata(int fd, struct t_unitdata *unitdata, struct t_iovec     *iov, unsigned int iovcount, int *flags);</pre>
DESCRIPTION	This function is used in connectionless mode to receive a data unit from another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be received, <i>unitdata</i> holds information associated with the received data unit, <i>iovcount</i> contains the number of non-contiguous udata buffers which is limited to $T\_IOV\_MAX$ , which is an implementation-defined value of at least 16, and <i>flags</i> is set on return to indicate that the complete data unit was not received. If the limit on <i>iovcount</i> is exceeded, the function fails with TBADDATA. The argument <i>unitdata</i> points to a t_unitdata structure containing the following members:
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; The maxlen field of addr and opt must be set before calling this function to indicate the maximum size of the buffer for each. The udata field of t_unitdata is not used. The iov_len and iov_base fields of "iov0" through iov [iovcount-1] must be set before calling t_rcvvudata() to define the buffer where the userdata will be placed. If the maxlen field of addr or opt is set to zero then no information is returned in the buf field for this parameter.</pre>
	On return from this call, <i>addr</i> specifies the protocol address of the sending user, <i>opt</i> identifies options that were associated with this data unit, and <i>iov</i> [0]. <i>iov_base</i> through <i>iov</i> [ <i>iovcount-1</i> ]. <i>iov_base</i> contains the user data that was received. The return value of t_rcvvudata() is the number of bytes of user data given to the user.
	Note that the limit on the total number of bytes available in all buffers passed:
	<i>iov</i> (0). <i>iov_len</i> + + <i>iov</i> ( <i>iovcount-1</i> ). <i>iov_len</i> ) may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.
	By default, t_rcvvudata() operates in synchronous mode and waits for a data unit to arrive if none is currently available. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcntl(2), t_rcvvudata() executes in asynchronous mode and fails if no data units are available.
	If the buffers defined in the <i>iov[]</i> array are not large enough to hold the current data unit, the buffers will be filled and T_MORE will be set in flags on return to indicate that another t_rcvvudata() should be called to retrieve the rest of the data unit. Subsequent calls to t_rcvvudata() will return zero for the length of the address and options, until the full data unit has been received.
	Networking Library Functions 653

t_rcvvudata(3NSL)			
<b>RETURN VALUES</b>	On successful completion, t_rcvvudata() returns the number of bytes received. Otherwise, it returns -1 on failure and t_errno is set to indicate the error.		
VALID STATES	T_IDLE.		
ERRORS	On failure, t_err	rno is set to one of the fol	lowing:
	TBADDATA	<i>iovcount</i> is greater than	T_IOV_MAX.
	TBADF	The specified file descri	ptor does not refer to a transport endpoint.
	TBUFOVFLW	or options ( <i>maxlen</i> ) is g	located for the incoming protocol address reater than 0 but not sufficient to store the lata information to be returned in <i>unitdata</i>
	TLOOK	An asynchronous event and requires immediate	t has occurred on this transport endpoint e attention.
	TNODATA	O_NONBLOCK was set, b from the transport prov	out no data units are currently available rider.
	TNOTSUPPORT	This function is not sup provider.	pported by the underlying transport
	TOUTSTATE	The communications er states in which a call to	ndpoint referenced by <i>fd</i> is not in one of the this function is valid.
	TPROTO		t a communication problem has been and the transport provider for which there is cror (t_errno).
	TSYSERR	A system error has occu	arred during execution of this function.
TLI	In the TLI interface definition, no counterpart of this routine was defined.		
COMPATIBILITY ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	TTA	TRIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe
SEE ALSO		oc(3NSL),t_open(3NSL) SL),t_sndudata(3NSL),	),t_rcvudata(3NSL), t_sndvudata(3NSL),attributes(5)

NAME	t_snd – send data or expedited data over a connection		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_snd(int fd, void *buf, unsigned int nbytes, int flags);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	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:		
	T_EXPEDITED	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.	
	T_MORE	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).	
		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.	
		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.	
	T_PUSH	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.	
	I		

t_snd(3NSL)	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 <i>info</i> 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:		
	<ul> <li>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 <i>info</i> argument.</li> <li>A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider.</li> </ul>		

		<ul> <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>
	TBADF	The specified file descriptor does not refer to a transport endpoint.
	TBADFLAG	An invalid flag was specified.
	TFLOW	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
	TLOOK	An asynchronous event has occurred on this transport endpoint.
	TNOTSUPPORT	This function is not supported by the underlying 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 (t_errno).
	TSYSERR	A system error has occurred during execution of this function.
TLI COMPATIBILITY	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.	
Interface Header	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< th=""><th>&gt;</th></tiuser.h<>	>
Error Description Values	The t_errno values that can be set by the XTI interface and cannot be set by the TLI interface are:	
	TPROTO TLOOK TBADFLAG TOUTSTATE The t_errno values that this routine can return under different circumstances than its XTI counterpart are: TBADDATA	

Networking Library Functions 657

t\_snd(3NSL)

	In the TBADDATA error cases described abo zero byte TSDU ( ETSDU) send attempts.	ove, TBADDATA is returned, only for illegal	
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT Level	Safe	
SEE ALSO	<pre>fcntl(2), t_getinfo(3NSL), t_look(3N attributes(5)</pre>	ISL),t_open(3NSL),t_rcv(3NSL),	
WARNINGS	It is important to remember that the transp endpoint as a single user. Therefore if seve calls then the different data may be interm	ral processes issue concurrent t_snd()	
	Multiple sends which exceed the maximum discovered by XTI. In this case an implement generated by the transport provider, perhaticate the form of a connection abort, a TSY.	entation-dependent error will result, ups on a subsequent XTI call. This error may	
	If multiple sends which exceed the maxim XTI, t_snd() fails with TBADDATA.	um TSDU or ETSDU size are detected by	

NAME	t_snddis - send user-initiated disconnection request		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_snddis(int fd, const struct t_call *call);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	This function is used to initiate an abortive release on an already established connection, or to reject a connection request. The argument <i>fd</i> identifies the local transport endpoint of the connection, and <i>call</i> specifies information associated with the abortive release. The argument <i>call</i> points to a t_call structure which contains the following members:		
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;The values in <i>call</i> have different semantics, depending on the context of the call to t_snddis(). When rejecting a connection request, <i>call</i> must be non-null and contain a valid value of <i>sequence</i> to uniquely identify the rejected connection indication to the transport provider. The <i>sequence</i> field is only meaningful if the transport connection is in the T_INCON state. The <i>addr</i> and <i>opt</i> fields of <i>call</i> are ignored. In all other cases, <i>call</i> need only be used when data is being sent with the disconnection request. The <i>addr</i>, <i>opt</i> and <i>sequence</i> fields of the t_call structure are ignored. If the user does not wish to send data to the remote user, the value of <i>call</i> may be a null pointer.</pre>		
	The <i>udata</i> structure specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider, as returned in the <i>discon</i> field, of the <i>info</i> argument of t_open(3NSL) or t_getinfo(3NSL). If the <i>len</i> field of <i>udata</i> is zero, no data will be sent to the remote user.		
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_DATAXFER, T_OUTCON, T_OUTREL, T_INREL, T_INCON(ocnt > 0).		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider.	
	TBADSEQ	An invalid sequence number was specified, or a null <i>call</i> pointer was specified, when rejecting a connection request.	

t\_snddis(3NSL)

TLOOK	An asynchronous event	, which requires attention, has occurred.
TNOTSUPPORT	This function is not sup provider.	ported by the underlying transport
TOUTSTATE	The communications er states in which a call to	ndpoint referenced by <i>fd</i> is not in one of the this function is valid.
TPROTO		t a communication problem has been nd the transport provider for which there is ror (t_errno).
TSYSERR	A system error has occu	urred during execution of this function.
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.		
The XTI interfaces use the header file, xti.h. TLI interfaces should <i>not</i> use this header. They should use the header:		
#include <tius< th=""><th>ser.h&gt;</th><th></th></tius<>	ser.h>	
The t_errno value TPROTO can be set by the XTI interface but not by the TLI interface.		
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.		
See attributes(	(5) for descriptions of the	following attributes:
ATT	RIBUTE TYPE	ATTRIBUTE VALUE
MT Level		Safe
<pre>t_connect(3NSL), t_getinfo(3NSL), t_listen(3NSL), t_open(3NSL), t_snd(3NSL), attributes(5) t_snddis() is an abortive disconnection. Therefore a t_snddis() issued on a connection endpoint may cause data previously sent by means of t_snd(3NSL), or data not yet received, to be lost, even if an error is returned.</pre>		
	TOUTSTATE TPROTO TSYSERR The XTI and TLI if files. This, and oth the subsections be The XTI interfaces header. They show #include <tius The t_errno value interface. The format of the the XTI interface, and See attributes( MT Level t_connect(3NSL), attributes()</tius 	provider.TOUTSTATEThe communications er states in which a call toTPROTOThis error indicates that detected between XTI a no other suitable XTI erTSYSERRA system error has occuThe XTI and TLI interface definitions have files. This, and other semantic differences b the subsections below.The XTI interfaces use the header file, xti. header. They should use the header:#include <tiuser.h>The t_errno value TPROTO can be set by to interface.The format of the options in an opt buffer the XTI interface, the TLI interface does not See attributes(5) for descriptions of theMT Levelt_connect(3NSL), t_getinfo(3NSL), t_ t_snd(3NSL), attributes(5)</tiuser.h>

NAME	t_sndrel – initiate an orderly release		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_sndrel(int fd);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	For transport providers of type T_COTS_ORD, this function is used to initiate an orderly release of the outgoing direction of data transfer and indicates to the transport provider that the transport user has no more data to send. The argument <i>fd</i> identifies the local transport endpoint where the connection exists. After calling t_sndrel(), the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has not been received. For transport providers of types other than T_COTS_ORD, this function fails with error TNOTSUPPORT.		
RETURN VALUES		completion, a value of 0 is returned. Otherwise, a value of –1 is _errno is set to indicate an error.	
VALID STATES	T_DATAXFER, T_INREL.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TFLOW	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.	
	TLOOK	An asynchronous event has occurred on this transport endpoint and requires immediate attention.	
	TNOTSUPPORT	This function is not supported by the underlying 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 (t_errno).	
	TSYSERR	A system error has occurred during execution of this function.	
TLI COMPATIBILITY		nterface definitions have common names but use different header ner semantic differences between the two interfaces are described in elow.	

## t\_sndrel(3NSL)

Interface Header	The XTI interfaces use the header file, xti. header. They should use the header:	h. TLI interfaces should <i>not</i> use this
	#include <tiuser.h></tiuser.h>	
Error Description Values	The t_errno values that can be set by the interface are:	XTI interface and cannot be set by the TLI
	TPROTO TLOOK TOUTSTATE	
Notes	Whenever this function fails with t_error been set.	set to TFLOW, O_NONBLOCK must have
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:
	ATTRIBUTE TYPE	ATTRIBUTE VALUE
	MT Level	Safe
SEE ALSO	t_error(3NSL),t_getinfo(3NSL),t_op attributes(5)	en(3NSL),t_rcvrel(3NSL),

## t\_sndreldata(3NSL)

NAME	t_sndreldata – init	iate or respond to an orderly release with user data	
SYNOPSIS	#include <xti.h></xti.h>		
	int <b>t_sndrelda</b>	<pre>ta(int fd, struct t_discon *discon);</pre>	
DESCRIPTION	transfer and to sen transport endpoin	ed to initiate an orderly release of the outgoing direction of data ad user data with the release. The argument <i>fd</i> identifies the local t where the connection exists, and <i>discon</i> points to a t_discon ag the following members:	
	over the connectio	ta; r calling t_sndreldata(), the user may not send any more data n. However, a user may continue to receive data if an orderly has not been received.	
	protocol-depender	ecifies the reason for the disconnection through a nt <i>reason code</i> , and <i>udata</i> identifies any user data that is sent with the field <i>sequence</i> is not used.	
	The <i>udata</i> structure specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider, as returned in the <i>discon</i> field of the <i>info</i> argument of t_open(3NSL) or t_getinfo(3NSL). If the <i>len</i> field of <i>udata</i> is zero or if the provider did not return T_ORDRELDATA in the t_open(3NSL) flags, no data will be sent to the remote user.		
	If a user does not wish to send data and reason code to the remote user, the value of <i>discon</i> may be a null pointer.		
	providers of servic	optional service of the transport provider, only supported by the type T_COTS_ORD. The flag T_ORDRELDATA in the <i>info→flag</i> to t_getinfo(3NSL) indicates that the provider elease user data.	
	This function may	not be available on all systems.	
RETURN VALUES		ompletion, a value of 0 is returned. Otherwise, a value of –1 is crno is set to indicate an error.	
VALID STATES	T_DATAXFER, T_INREL.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADDATA	The amount of user data specified was not within the bounds allowed by the transport provider, or user data was supplied and the provider did not return T_ORDRELDATA in the t_open(3NSL) flags.	
	TBADF	The specified file descriptor does not refer to a transport endpoint.	

t\_sndreldata(3NSL)

	TFLOW	—	put the flow control mechanism prevented from accepting the function at this time.
	TLOOK	An asynchronous event has occurred on this transport endpoi and requires immediate attention.	
	TNOTSUPPORT	Orderly release is not su provider.	apported by the underlying transport
	TOUTSTATE	The communications er states in which a call to	adpoint referenced by $fd$ is not in one of the this function is valid.
	TPROTO		t a communication problem has been nd the transport provider for which there is ror (t_errno).
	TSYSERR	A system error has occu	rred during execution of this function.
TLI	In the TLI interfac	e definition, no counterpa	art of this routine was defined.
COMPATIBILITY ATTRIBUTES	See attributes(	5) for descriptions of the	following attributes:
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe
	t_getinfo(3NSL), t_open(3NSL), t_rcvrel(3NSL), t_rcvreldata(3NSL), t_sndrel(3NSL), attributes(5)		
SEE ALSO			rrel(3NSL),t_rcvreldata(3NSL),
SEE ALSO NOTES	t_sndrel(3NSL), The interfaces t_s specific transport	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	rrel(3NSL), t_rcvreldata(3NSL), cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform. conjunction with Internet Transports (TCP
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.
	t_sndrel(3NSL), The interfaces t_s specific transport of These interfaces and	, attributes(5) endreldata() and t_ro called "minimal OSI," wh	cvreldata(3NSL) are only for use with a nich is not available on the Solaris platform.

## t\_sndudata(3NSL)

SYNOPSIS       #include <xti.h>         int t_sndudata (int fd, const struct t_unitdata *unitdata);         DESCRIPTION       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</xti.h>	
<b>DESCRIPTION</b> 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	
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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.	
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 t_unitdata structure containing the following members:	
struct netbuf addr; struct netbuf opt; struct netbuf udata; In <i>unitdata, 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.	
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 t_sndudata() will return -1 with t_errno set to TBADDATA.	1
By default, t_sndudata() operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NONBLOCK is set by means of t_open(3NSL) or fcnt1(2), t_sndudata() 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 t_look(3NSL) or the EM interface.	t
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 t_open(3NSL) or t_getinfo(3NSL), a TBADDATA error will be generated. If t_sndudata() is called before the destination user has activate its transport endpoint (see t_bind(3NSL)), the data unit may be discarded.	r
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 $t_{rcvuderr}$ . Therefore, an application must be prepared to receive these errors in both of these ways.	
If the call is interrupted, t_sndudata() will return EINTR and the datagram will no be sent.	ot

t_sndudata(3NSL)			
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of $-1$ is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADADDR	The specified protocol address was in an incorrect format or contained illegal information.	
	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.	
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TBADOPT	The specified options were in an incorrect format or contained illegal information.	
	TFLOW	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.	
	TLOOK	An asynchronous event has occurred on this transport endpoint.	
	TNOTSUPPORT	This function is not supported by the underlying 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 (t_errno).	
	TSYSERR	A system error has occurred during execution of this function.	
TLI COMPATIBILITY		nterface definitions have common names but use different header er semantic differences between the two interfaces are described in low.	
Interface Header	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:< th=""><th>&gt;</th></tiuser.h:<>	>	
Error Description Values	The t_errno valu interface are:	ues that can be set by the XTI interface and cannot be set by the TLI	
	TPROTO TBADADDR TBADOPT		

666 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

## t\_sndudata(3NSL)

	TLOOK TOUTSTATE		
Notes	Whenever this function fails with t_error set to $\tt TFLOW, O\_NONBLOCK$ must have been set.		
<b>Option Buffers</b>	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.		
ATTRIBUTES	See attributes(5) for descriptions of the	following attributes:	
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT Level	Safe	
SEE ALSO	<pre>fcntl(2), t_alloc(3NSL), t_bind(3NSL) t_look(3NSL), t_open(3NSL), t_rcvuda attributes(5)</pre>		
		Networking Library Functions 667	

## t\_sndv(3NSL)

NAME	t_sndv – send data connection	or expedited data, from one or more non-contiguous buffers, on a	
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_sndv(int</pre>	<pre>fd, const struct t_iovec *iov, unsigned int iovcount,</pre>	
DESCRIPTION	identifies the local array of buffer add <i>iov0</i> , <i>iov1</i> , throug data buffers which	ed to send either normal or expedited data. The argument <i>fd</i> transport endpoint over which data should be sent, <i>iov</i> points to an lress/buffer length pairs. t_sndv() sends data contained in buffers h <i>iov</i> [ <i>iovcount-1</i> ]. <i>iovcount</i> contains the number of non-contiguous is limited to T_IOV_MAX, an implementation-defined value of at t is exceeded, the function fails with TBADDATA.	
	$iov(0).iov\_len + + iov(i)$	ovcount-1).iov_len)	
	Note that the limit	on the total number of bytes available in all buffers passed:	
	may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.		
	The argument <i>flags</i> specifies any optional flags described below:		
	T_EXPEDITED	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.	
	T_MORE	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_sndv() calls. Each t_sndv() with the T_MORE flag set indicates that another t_sndv() or t_snd(3NSL) will follow with more data for the current TSDU (or ETSDU).	
	flag not set. Use of losing the boundar implies nothing ab interface. If the tra indicated in the <i>inj</i>	DU (or ETSDU) is identified by a t_sndv() call with the T_MORE T_MORE enables a user to break up large logical data units without ries of those units at the other end of the connection. The flag bout how the data is packaged for transfer below the transport nsport provider does not support the concept of a TSDU as to argument on return from t_open(3NSL) or t_getinfo(3NSL), not meaningful and will be ignored if set.	
	this is used to indi	ero-length fragment of a TSDU or ETSDU is only permitted where cate the end of a TSDU or ETSDU, that is, when the T_MORE flag is sport providers also forbid zero-length TSDUs and ETSDUs.	
	not sent. The reque	ests that the provider transmit all data that it has accumulated but est is a local action on the provider and does not affect any similarly ag (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.
	The communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.
	By default, t_sndv() 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_sndv() executes 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 via either t_look(3NSL) or the EM interface.
	On successful completion, t_sndv() returns the number of bytes accepted by the transport provider. Normally this will equal the total number of bytes to be sent, that is,
	( <i>iov</i> 0.iov_len + + iov[iovcount-1].iov_len)
	However, the interface is constrained to send at most INT_MAX bytes in a single send. When t_sndv() has submitted INT_MAX (or lower constrained value, see the note above) bytes to the provider for a single call, this value is returned to the user. 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_sndv() returns a value that is less than the value of nbytes. If t_sndv() 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 the number of bytes of data in the <i>iov</i> array is zero and sending of zero octets is not supported by the underlying transport service, t_sndv() 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 <i>info</i> 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_sndv() 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 in synchronous mode, if more than INT_MAX bytes of data are passed in the <i>iov</i> array, only the first INT_MAX bytes will be passed to the provider.
	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

t_sndv(3NSL)			
	communications provider is blocked due to flow control, or that O_NONBLOCK is clear and the function was interrupted by a signal.		
VALID STATES	T_DATAXFER, T_1	T_DATAXFER, T_INREL.	
ERRORS	On failure, t_errno is set to one of the following:		
	TBADDATA	Illegal amount of data:	
	TBADF	The specified file description	ptor does not refer to a transport endpoint.
		<ul> <li>fragment TSDU (ETS current values of the argument.</li> <li>A send of a zero byte TSDU (ETSDU) is not Multiple sends were larger than that spec ETSDU fields in the implementation to d</li> </ul>	tempted specifying a TSDU (ETSDU) or SDU) greater than that specified by the TSDU or ETSDU fields in the <i>info</i> e TSDU (ETSDU) or zero byte fragment of a ot supported by the provider. attempted resulting in a TSDU (ETSDU) ified by the current value of the TSDU or <i>info</i> argument – the ability of an XTI etect such an error case is endent. See WARNINGS, below. an T_IOV_MAX.
	TBADFLAG	An invalid flag was spe	cified.
	TFLOW		ut the flow control mechanism prevented rom accepting any data at this time.
	TLOOK	An asynchronous event	has occurred on this transport endpoint.
	TNOTSUPPORT	This function is not sup provider.	ported by the underlying transport
	TOUTSTATE	The communications en states in which a call to	dpoint referenced by <i>fd</i> is not in one of the this function is valid.
	TPROTO		a communication problem has been nd the transport provider for which there is ror (t_errno).
	TSYSERR	A system error has occu	rred during execution of this function.
TLI COMPATIBILITY ATTRIBUTES	In the TLI interface definition, no counterpart of this routine was defined. See attributes(5) for descriptions of the following attributes:		
	ATT	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe

670 man pages section 3: Networking Library Functions • Last Revised 23 Aug 2001

- SEE ALSO | t\_getinfo(3NSL), t\_open(3NSL), t\_rcvv(3NSL) t\_rcv(3NSL), t\_snd(3NSL), attributes(5)
- **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\_sndv() or t\_snd(3NSL) 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 TSYSERR, a TBADDATA or a TPROTO error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, t sndv() fails with TBADDATA.

t_sndvudata(3NSL)	
NAME	t_sndvudata – send a data unit from one or more noncontiguous buffers
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>
	<pre>int t_sndvudata(int fd, struct t_unitdata *unitdata, struct t_iovec     *iov, unsigned int iovcount);</pre>
DESCRIPTION	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, <i>iovcount</i> contains the number of non-contiguous <i>udata</i> buffers and is limited to an implementation-defined value given by T_IOV_MAX which is at least 16, and <i>unitdata</i> points to a t_unitdata structure containing the following members:
	<pre>struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre>
	If the limit on <i>iovcount</i> is exceeded, the function fails with TBADDATA.
	In unitdata, <i>addr</i> specifies the protocol address of the destination user, and <i>opt</i> identifies options that the user wants associated with this request. The <i>udata</i> field is not used. 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 may use default options.
	The data to be sent is identified by <i>iov</i> [0] through <i>iov</i> [ <i>iovcount-1</i> ].
	Note that the limit on the total number of bytes available in all buffers passed:
	iov(0).iov_len + + iov(iovcount-1).iov_len
	may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.
	By default, t_sndvudata() 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_sndvudata() executes 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 t_look(3NSL) or the EM interface.
	If the amount of data specified in <i>iov</i> 0 through <i>iov</i> [ <i>iovcount-1</i> ] exceeds the TSDU size as returned in the <i>tsdu</i> field of the <i>info</i> argument of t_open(3NSL) or t_getinfo(3NSL), or is zero and sending of zero octets is not supported by the underlying transport service, a TBADDATA error is generated. If t_sndvudata() is called before the destination user has activated its transport endpoint (see t_bind(3NSL)), the data unit may be discarded.

	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 t_rcvuderr(3NSL). An application must therefore be prepared to receive these errors in both of these ways.		
<b>RETURN VALUES</b>	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE.		
ERRORS	On failure, t_err	rno is set to one of the foll	owing:
	TBADADDR	The specified protocol a contained illegal inform	uddress was in an incorrect format or nation.
	TBADDATA	Illegal amount of data.	
	<ul> <li>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.</li> <li><i>iovcount</i> is greater than T IOV MAX.</li> </ul>		
	TBADF         The specified file descriptor does not refer to a transport endpoint		ptor does not refer to a transport endpoint.
	TBADOPTThe specified options were in an incorrect format or contained illegal information.		
	TFLOWO_NONBLOCK i was set, but the flow control mechanism preven the transport provider from accepting any data at this time.		-
	TLOOK	An asynchronous event	has occurred on this transport endpoint.
	TNOTSUPPORT This function is not supported by the underlying transport provider.		ported by the underlying transport
	TOUTSTATE	The communications er states in which a call to	idpoint referenced by <i>fd</i> is not in one of the this function is valid.
	TPROTO		t a communication problem has been nd the transport provider for which there is ror (t_errno).
	TSYSERR	A system error has occu	rred during execution of this function.
TLI			art of this routine was defined.
COMPATIBILITY ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		following attributes:
	TTA	RIBUTE TYPE	ATTRIBUTE VALUE
	MT Level		Safe
	L		

t_sndvudata(3NSL)			
SEE ALSO	fcntl(2),t alloc(3NSL),t open(3NSL),t rcvudata(3NSL),		
	t_rcvvudata(3NSL)t_rcvuderr(3NSL),t_sndudata(3NSL),attributes(5)		

NAME	t_strerror – produce an error message string		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>const char *t_strerror(int errnum);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	The t_strerror() function maps the error number in <i>errnum</i> that corresponds to an XTI error to a language-dependent error message string and returns a pointer to the string. The string pointed to will not be modified by the program, but may be overwritten by a subsequent call to the <i>t_strerror</i> function. The string is not terminated by a newline character. The language for error message strings written by t_strerror() is that of the current locale. If it is English, the error message string describing the value in t_errno may be derived from the comments following the t_errno codes defined in <xti.h>. If an error code is unknown, and the language is English, t_strerror() returns the string:</xti.h>		
	" <error>: error unknown"where <error> is the error number supplied as input. In other languages, an equivalent text is provided.</error></error>		
VALID STATES	ALL - apart from T_UNINIT.		
<b>RETURN VALUES</b>	The function t_strerror() returns a pointer to the generated message string.		
TLI COMPATIBILITY	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.		
Interface Header	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></tiuser.h>		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE ATTRIBUTE VALUE		
	MT Level Safe		
SEE ALSO	t_errno(3NSL),t_error(3NSL), attributes(5)		

## t\_sync(3NSL)

NAME	t_sync – synchroni	ize transport library	
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_sync(int fd);</pre>		
DESCRIPTION	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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	For the transport endpoint specified by <i>fd</i> , t_sync() synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, it can convert an uninitialized file descriptor (obtained by means of a open(2), dup(2) or as a result of a fork(2) and exec(2)) to an initialized transport endpoint, assuming that the file descriptor referenced a transport endpoint, by updating and allocating the necessary library data structures. This function also allows two cooperating processes to synchronize their interaction with a transport provider.		
	For example, if a process forks a new process and issues an $exec(2)$ , the new process must issue a t_sync() to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.		
	It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activities so as not to violate the state of the transport endpoint. The function t_sync() returns the current state of the transport endpoint to the user, thereby enabling the user to verify the state before taking further action. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event could change the endpoint's state <i>after</i> a t_sync() is issued.		
	If the transport endpoint is undergoing a state transition when $t_sync()$ is called, the function will fail.		
RETURN VALUES	On successful completion, the state of the transport endpoint is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error. The state returned is one of the following:		
	T_UNBND	Unbound.	
	T_IDLE	Idle.	
	T_OUTCON	Outgoing connection pending.	
	T_INCONIncoming connection pending.		
	T_DATAXFER	Data transfer.	
	1		

676 man pages section 3: Networking Library Functions • Last Revised 7 May 1998

T_INRELIncoming orderly release (waiting for an orderly release request).ERRORSOn failure, t_errmo is set to one of the following: TBADFThe specified file descriptor does not refer to a transport endpoint. This error may be returned when the <i>fd</i> has been previously closed or an erroneous number may have been passed to the call.TPROTOThe seror indicates that a communication problem has been detected between XII and the transport provider for which there is no other suitable XII error (t_errno).TSTATECHNGThe transport endpoint is undergoing a state change. TSYSERRA system error has occurred during execution of this function.The XII 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.Interface HeaderThey should use the header: #include <tiuser.h>#include <tiuser.h>TPROTOTPROTOATTRIBUTESSee attributes(5) for descriptions of the following attributes:SEE ALSOdup(2), exec(2), fork(2), open(2), attributes(5)</tiuser.h></tiuser.h>		T_OUTREL	Outgoing orderly releas indication).	se (waiting for an orderly release
TBADF       The specified file descriptor does not refer to a transport endpoint. This error may be returned when the <i>fd</i> has been previously closed or an erroneous number may have been passed to the call.         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).         TSTATECHNG       The transport endpoint is undergoing a state change.         TSYSERR       A system error has occurred during execution of this function.         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.         Interface Header       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>       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO       See attributes(5) for descriptions of the following attributes:         ITPROTO       ATTRIBUTE YPE         ATTRIBUTE VALUE       MT Level</tiuser.h>		T_INREL Incoming orderly release (waiting for an orderly release		
This error may be returned when the fd has been previously closed or an erroneous number may have been passed to the call.         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).         TSTATECHNG       The transport endpoint is undergoing a state change.         TSYSERR       A system error has occurred during execution of this function.         TLI       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.         Interface Header       The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:         #include <tiuser.h>       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO       See attributes(5) for descriptions of the following attributes:         MTLEVEL       ATTRIBUTE Stafe</tiuser.h>	ERRORS	On failure, t_errno is set to one of the following:		
detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).         TSTATECHNG       The transport endpoint is undergoing a state change.         TSYSERR       A system error has occurred during execution of this function.         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.         Interface Header       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>       #include <tiuser.h>         TPROTO       See attributes(5) for descriptions of the following attributes:         MTLEVEN       ATTRIBUTE VALUE         MT Level       Safe</tiuser.h></tiuser.h>		TBADF	This error may be retur	ned when the <i>fd</i> has been previously closed
TSYSER       A system error has occurred during execution of this function.         TLI       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.         Interface Header       The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:         #include <tiuser.h>       #include <tiuser.h>         Error Description Values       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO       See attributes(5) for descriptions of the following attributes:         MT Level       Attribute Value</tiuser.h></tiuser.h>		TPROTO	detected between XTI a	nd the transport provider for which there is
TLI COMPATIBILITY       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.         Interface Header       The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:         #include <tuser.h>       #include <tuser.h>         Error Description Values       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO       See attributes(5) for descriptions of the following attributes:         MT Level       ATTRIBUTE VALUE</tuser.h></tuser.h>		TSTATECHNG	The transport endpoint	is undergoing a state change.
COMPATIBILITY       files. This, and other semantic differences between the two interfaces are described in the subsections below.         Interface Header       The XTI interfaces use the header file, xti.h. TLI interfaces should not use this header. They should use the header:         #include <tiuser.h>       #include <tiuser.h>         Error Description Values       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO       See attributes(5) for descriptions of the following attributes:         MT Level       Safe</tiuser.h></tiuser.h>		TSYSERR	A system error has occu	urred during execution of this function.
header. They should use the header:         #include <tiuser.h>         #include <tiuser.h>         The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         TPROTO         See attributes(5) for descriptions of the following attributes:         Image: the total interface is in the total interface is in</tiuser.h></tiuser.h>		files. This, and other semantic differences between the two interfaces are described in		
Error Description Values       The t_errno value that can be set by the XTI interface and cannot be set by the TLI interface is:         ATTRIBUTES       TPROTO         ATTRIBUTES       See attributes(5) for descriptions of the following attributes:         MT Level       Safe	Interface Header			
Values       interface is:         TPROTO         ATTRIBUTES       See attributes(5) for descriptions of the following attributes:         ATTRIBUTE TYPE       ATTRIBUTE VALUE         MT Level       Safe		#include <tiuser.h< th=""><th>&gt;</th><th></th></tiuser.h<>	>	
ATTRIBUTES       See attributes(5) for descriptions of the following attributes:         ATTRIBUTE TYPE       ATTRIBUTE VALUE         MT Level       Safe	Error Description Values			
ATTRIBUTE TYPE     ATTRIBUTE VALUE       MT Level     Safe		TPROTO		
MT Level Safe	ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
MT Level Safe				
		ATTI	RIBUTE TYPE	ATTRIBUTE VALUE
SEE ALSO dup(2), exec(2), fork(2), open(2), attributes(5)		MT Level		Safe
	SEE ALSO	dup(2), exec(2), f	ork(2), open(2), attrik	outes(5)

t_sysconf(3NSL)			
NAME	t_sysconf – get configurable XTI variables		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_sysconf(intname);</pre>		
DESCRIPTION	The t_sysconf() function provides a met current value of configurable and implement		
	The <i>name</i> argument represents the XTI syste table lists the minimal set of XTI system var by t_sysconf(), and the symbolic consta corresponding values used for <i>name</i> .	riables from <xti.h> that can be returned</xti.h>	
	Variable	Value of Name	
	T_IOV_MAX	_SC_T_IOV_MAX	
RETURN VALUES	If <i>name</i> is valid, t_sysconf() returns the value of the requested limit/option, which might be -1, and leaves t_errno unchanged. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	All.		
ERRORS	On failure, t_errno is set to the following:		
	TBADFLAG <i>name</i> has an invalid value.		
TLI	In the TLI interface definition, no counterpart of this routine was defined.		
COMPATIBILITY ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	MT-Safe	
SEE ALSO	sysconf(3C), t_rcvv(3NSL), t_rcvvuda t_sndvudata(3NSL), attributes(5)	ta(3NSL), t_sndv(3NSL),	

NAME	t_unbind – disable a transport endpoint		
SYNOPSIS	<pre>#include <xti.h></xti.h></pre>		
	<pre>int t_unbind(int fd);</pre>		
DESCRIPTION	The 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 tiuser.h header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.		
	t_unbind() function disables the transport endpoint specified by <i>fd</i> which was previously bound by t_bind(3NSL). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider. An endpoint which is disabled by using t_unbind() can be enabled by a subsequent call to t_bind(3NSL).		
RETURN VALUES	Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.		
VALID STATES	T_IDLE.		
ERRORS	On failure, t_errno is set to one of the following:		
	TBADF	The specified file descriptor does not refer to a transport endpoint.	
	TLOOK	An asynchronous event has occurred on this transport endpoint.	
	TOUTSTATE	The communications endpoint referenced by $fd$ 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 (t_errno).	
	TSYSERR	A system error has occurred during execution of this function.	
TLI COMPATIBILITY	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.		
Interface Header	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:< th=""><th>&gt;</th></tiuser.h:<>	>	
Error Description Values	The t_errno valu interface is:	e that can be set by the XTI interface and cannot be set by the TLI	
	TPROTO		

Networking Library Functions 679

#### t\_unbind(3NSL)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

# SEE ALSO t\_bind(3NSL), attributes(5)

NAME	xdr – library routines for external data representation	
DESCRIPTION	XDR routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls (RPC) are transmitted using these routines.	
Index to Routines	The following table lists XDR routines an are described:	d the manual reference pages on which they
	XDR Routine	Manual Reference Page
	xdr_array	xdr_complex(3NSL)
	xdr_bool	xdr_simple(3NSL)
	xdr_bytes	xdr_complex(3NSL)
	xdr_char	xdr_simple(3NSL)
	xdr_control	xdr_admin(3NSL)
	xdr_destroy	xdr_create(3NSL)
	xdr_double	xdr_simple(3NSL)
	xdr_enum	xdr_simple(3NSL)
	xdr_float	xdr_simple(3NSL)
	xdr_free	xdr_simple(3NSL)
	xdr_getpos	xdr_admin(3NSL)
	xdr_hyper	xdr_simple(3NSL)
	xdr_inline	xdr_admin(3NSL)
	xdr_int	xdr_simple(3NSL)
	xdr_long	xdr_simple(3NSL)
	xdr_longlong_t	xdr_simple(3NSL)
	xdr_opaque	xdr_complex(3NSL)
	xdr_pointer	xdr_complex(3NSL)
	xdr_quadruple	xdr_simple(3NSL)
	xdr_reference	xdr_complex(3NSL)
	xdr_setpos	xdr_admin(3NSL)
	xdr_short	xdr_simple(3NSL)
	xdr_sizeof	xdr_admin(3NSL)
	xdr_string	xdr_complex(3NSL)
	xdr_u_char	xdr_simple(3NSL)

xdr(3NSL)

xdr_u_hyper	xdr_simple(3NSL)
xdr_u_int	xdr_simple(3NSL)
xdr_u_long	xdr_simple(3NSL)
xdr_u_longlong_t	xdr_simple(3NSL)
xdr_u_short	xdr_simple(3NSL)
xdr_union	$xdr_complex(3NSL)$
xdr_vector	xdr_complex(3NSL)
xdr_void	xdr_simple(3NSL)
xdr_wrapstring	xdr_complex(3NSL)
xdrmem_create	xdr_create(3NSL)
xdrrec_create	xdr_create(3NSL)
xdrrec_endofrecord	xdr_admin(3NSL)
xdrrec_eof	xdr_admin(3NSL)
xdrrec_readbytes	xdr_admin(3NSL)
xdrrec_skiprecord	xdr_admin(3NSL)
xdrstdio_create	xdr_create(3NSL)

#### ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

SEE ALSO rpc(3NSL), xdr\_admin(3NSL), xdr\_complex(3NSL), xdr\_create(3NSL), xdr\_simple(3NSL), attributes(5)

NAME	xdr_admin, xdr_control, xdr_getpos, xdr_inline, xdrrec_endofrecord, xdrrec_eof, xdrrec_readbytes, xdrrec_skiprecord, xdr_setpos, xdr_sizeof – library routines for external data representation
DESCRIPTION	XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.
	These routines deal specifically with the management of the XDR stream.
Routines	See rpc(3NSL) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested either that malloc(3C) be used to allocate these buffers, or that the programmer insure that the buffer address is divisible evenly by four.
	<pre>#include <rpc xdr.h=""></rpc></pre>
	<pre>bool_t xdr_control( XDR *xdrs, int req, void *info); A function macro to change or retrieve various information about an XDR stream. req indicates the type of operation and info is a pointer to the information. The supported values of req is XDR_GET_BYTES_AVAIL and its argument type is xdr_bytesrec *. They return the number of bytes left unconsumed in the stream and a flag indicating whether or not this is the last fragment.</pre>
	<pre>uint_t xdr_getpos(const XDR *xdrs); A macro that invokes the get-position routine associated with the XDR stream, xdrs. The routine returns an unsigned integer, which indicates the position of the XDR byte stream. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances need not guarantee this. Therefore, applications written for portability should not depend on this feature.</pre>
	<pre>long *xdr_inline (XDR *xdrs, const int len); A macro that invokes the in-line routine associated with the XDR stream, xdrs. The routine returns a pointer to a contiguous piece of the stream's buffer; len is the byte length of the desired buffer. Note: pointer is cast to long *.</pre>
	Warning: xdr_inline() may return NULL (0) if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency, and applications written for portability should not depend on this feature.
	<pre>bool_t xdrrec_endofrecord(XDR *xdrs, int sendnow); This routine can be invoked only on streams created by xdrrec_create(). See xdr_create(3NSL). The data in the output buffer is marked as a completed record, and the output buffer is optionally written out if sendnow is non-zero. This routine returns TRUE if it succeeds, FALSE otherwise.</pre>
	<pre>bool_t xdrrec_eof(XDR *xdrs); This routine can be invoked only on streams created by xdrrec_create(). After consuming the rest of the current record in the stream, this routine returns TRUE if</pre>

xdr_admin(3NSL)
-----------------

	there is no more data in the stream's input buffer. It returns FALSE if there is additional data in the stream's input buffer.		
	<pre>int xdrrec_readbytes(XDR *xdrs, caddr_t addr, uint_t nbytes); This routine can be invoked only on streams created by xdrrec_create(). It attempts to read nbytes bytes from the XDR stream into the buffer pointed to by addr. Upon success this routine returns the number of bytes read. Upon failure, it returns -1. A return value of 0 indicates an end of record.</pre>		
	<pre>bool_t xdrrec_skiprecord(XDR *xdrs); This routine can be invoked only on streams created by xdrrec_create(). See xdr_create(3NSL). It tells the XDR implementation that the rest of the current record in the stream's input buffer should be discarded. This routine returns TRUE if it succeeds, FALSE otherwise.</pre>		
	<pre>bool_t xdr_setpos(XDR *xdrs, const uint_t pos); A macro that invokes the set position routine associated with the XDR stream xdrs. The parameter pos is a position value obtained from xdr_getpos(). This routine returns TRUE if the XDR stream was repositioned, and FALSE otherwise.</pre>		
	Warning: it is difficult to reposition some types of XDR streams, so this routine may fail with one type of stream and succeed with another. Therefore, applications written for portability should not depend on this feature.		
	<pre>unsigned long xdr_sizeof (xdrproc_t func, void *data); This routine returns the number of bytes required to encode data using the XDR filter function func, excluding potential overhead such as RPC headers or record markers. 0 is returned on error. This information might be used to select between transport protocols, or to determine the buffer size for various lower levels of RPC client and server creation routines, or to allocate storage when XDR is used outside of the RPC subsystem.</pre>		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	ATTRIBUTE TYPE	ATTRIBUTE VALUE	
	MT-Level	Safe	
SEE ALSO	<pre>malloc(3C), rpc(3NSL), xdr_complex(3NSL), xdr_create(3NSL), xdr_simple(3NSL), attributes(5)</pre>		

- NAME | xdr\_complex, xdr\_array, xdr\_bytes, xdr\_opaque, xdr\_pointer, xdr\_reference, xdr\_string, xdr\_union, xdr\_vector, xdr\_wrapstring – library routines for external data representation
- **DESCRIPTION** XDR library routines allow C programmers to describe complex data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data. These routines are the XDR library routines for complex data structures. They require the creation of XDR streams. See xdr create(3NSL).
  - **Routines** See rpc(3NSL) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested either that malloc() be used to allocate these buffers, or that the programmer insure that the buffer address is divisible evenly by four.

#include <rpc/xdr.h>

bool\_t xdr\_array(XDR \*xdrs, caddr\_t \*arrp, uint\_t \*sizep, const uint\_t maxsize, const uint\_t elsize, const xdrproc\_t elproc);

xdr\_array() translates between variable-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *sizep* is the address of the element count of the array; this element count cannot exceed *maxsize*. The parameter *elsize* is the size of each of the array's elements, and *elproc* is an XDR routine that translates between the array elements' C form and their external representation. If *\*aarp* is NULL when decoding, xdr\_array() allocates memory and *\*aarp* points to it. This routine returns TRUE if it succeeds, FALSE otherwise.

- bool\_t xdr\_bytes(XDR \*xdrs, char \*\*sp, uint\_t \*sizep, const uint\_t maxsize); xdr\_bytes() translates between counted byte strings and their external representations. The parameter sp is the address of the string pointer. The length of the string is located at address sizep; strings cannot be longer than maxsize. If \*sp is NULL when decoding, xdr\_bytes() allocates memory and \*sp points to it. This routine returns TRUE if it succeeds, FALSE otherwise.
- bool\_t xdr\_opaque(XDR \*xdrs, caddr\_t cp, const uint\_t cnt); xdr\_opaque() translates between fixed size opaque data and its external representation. The parameter cp is the address of the opaque object, and cnt is its size in bytes. This routine returns TRUE if it succeeds, FALSE otherwise.
- bool\_t xdr\_pointer(XDR \*xdrs, char \*\*objpp, uint\_t objsize, const xdrproc\_t xdrobj); Like xdr\_reference() except that it serializes null pointers, whereas xdr\_reference() does not. Thus, xdr\_pointer() can represent recursive data structures, such as binary trees or linked lists. If \*objpp is NULL when decoding, xdr\_pointer() allocates memory and \*objpp points to it.
- bool\_t xdr\_reference(XDR \*xdrs, caddr\_t \*pp, uint\_t size, const xdrproc\_t proc); xdr\_reference() provides pointer chasing within structures. The parameter pp is the address of the pointer; size is the sizeof the structure that \*pp points to; and proc is an XDR procedure that translates the structure between its C form and

#### xdr\_complex(3NSL)

its external representation. If \**pp* is NULL when decoding, xdr\_reference() allocates memory and \**pp* points to it. This routine returns 1 if it succeeds, 0 otherwise.

Warning: this routine does not understand null pointers. Use xdr\_pointer() instead.

bool\_t xdr\_string(XDR \*xdrs, char \*\*sp, const uint\_t maxsize);

xdr\_string() translates between C strings and their corresponding external representations. Strings cannot be longer than *maxsize*. Note: *sp* is the address of the string's pointer. If *\*sp* is NULL when decoding, xdr\_string() allocates memory and *\*sp* points to it. This routine returns TRUE if it succeeds, FALSE otherwise. Note: xdr\_string() can be used to send an empty string (" "), but not a null string.

bool\_t xdr\_union(XDR \*xdrs, enum\_t \*dscmp, char \*unp, const struct xdr\_discrim \*choices, const xdrproc\_t (\*defaultarm));

xdr\_union() translates between a discriminated C union and its corresponding external representation. It first translates the discriminant of the union located at *dscmp*. This discriminant is always an enum\_t. Next the union located at *unp* is translated. The parameter *choices* is a pointer to an array of xdr\_discrim structures. Each structure contains an ordered pair of [*value*, *proc*]. If the union's discriminant is equal to the associated *value*, then the proc is called to translate the union. The end of the xdr\_discrim structure array is denoted by a routine of value NULL. If the discriminant is not found in the *choices* array, then the *defaultarm* procedure is called (if it is not NULL). It returns TRUE if it succeeds, FALSE otherwise.

bool\_t xdr\_vector(XDR \*xdrs, char \*arrp, const uint\_t size, const uint\_t elsize, const xdrproc\_t elproc);

xdr\_vector() translates between fixed-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while size is the element count of the array. The parameter *elsize* is the sizeof each of the array's elements, and *elproc* is an XDR routine that translates between the array elements' C form and their external representation. This routine returns TRUE if it succeeds, FALSE otherwise.

bool\_t xdr\_wrapstring(XDR \*xdrs, char \*\*sp);

A routine that calls xdr\_string (*xdrs*, *sp*, *maxuint*); where *maxuint* is the maximum value of an unsigned integer.

Many routines, such as xdr\_array(), xdr\_pointer(), and xdr\_vector() take a function pointer of type xdrproc\_t(), which takes two arguments. xdr\_string(), one of the most frequently used routines, requires three arguments, while xdr\_wrapstring() only requires two. For these routines, xdr\_wrapstring() is desirable. This routine returns TRUE if it succeeds, FALSE otherwise.

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

xdr\_complex(3NSL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

# SEE ALSO malloc(3C), rpc(3NSL), xdr\_admin(3NSL), xdr\_create(3NSL), xdr\_simple(3NSL), attributes(5)

xdr\_create(3NSL)

NAME	xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation	
SYNOPSIS	<pre>#include <rpc xdr.h=""></rpc></pre>	
	<pre>void xdr_destroy(XDR *xdrs);</pre>	
	<pre>void xdrmem_create(XDR *xdrs, const caddr_t addr, const uint_t size,</pre>	
	<pre>void xdrrec_create(XDR *xdrs, const uint_t sendsz, const uint_t     recvsz, const caddr_t handle, const int (*readit)const void     *read_handle, char *buf, const int len, const int (*writeit)const     void *write_handle, const char *buf, const int len);</pre>	
	<pre>void xdrstdio_create(XDR *xdrs, FILE *file, const enum xdr_op op);</pre>	
DESCRIPTION	The XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.	
	These routines deal with the creation of XDR streams, which must be created before any data can be translated into XDR format.	
Routines	See rpc(3NSL) for the definition of the XDR CLIENT and SVCXPRT data structures. Any buffers passed to the XDR routines must be properly aligned. Use malloc(3C) to allocate these buffers or be sure that the buffer address is divisible evenly by four.	
	<pre>xdr_destroy() A macro that invokes the destroy routine associated with the XDR stream, xdrs. Private data structures associated with the stream are freed. Using xdrs after xdr_destroy() is invoked is undefined.</pre>	
	<pre>xdrmem_create() This routine initializes the XDR stream object pointed to by xdrs. The stream's data is written to or read from a chunk of memory at location addr whose length is no less than size bytes long. The op determines the direction of the XDR stream. The value of op can be either XDR_ENCODE, XDR_DECODE, or XDR_FREE.</pre>	
	<pre>xdrrec_create() This routine initializes the read-oriented XDR stream object pointed to by xdrs. The stream's data is written to a buffer of size sendsz. A value of 0 indicates the system should use a suitable default. The stream's data is read from a buffer of size recvsz. It too can be set to a suitable default by passing a 0 value. When a stream's output buffer is full, writeit is called. Similarly, when a stream's input buffer is empty, xdrrec_create() calls readit. The behavior of these two routines is similar to the system calls read() and write(), except that an appropriate handle, read_handle or write_handle, is passed to the former routines as the first parameter instead of a file descriptor. See read(2) and write(2), respectively. The XDR stream's op field must be set by the caller.</pre>	

This XDR stream implements an intermediate record stream. Therefore, additional bytes in the stream are provided for record boundary information.

xdrstdio create()

This routine initializes the XDR stream object pointed to by *xdrs*. The XDR stream data is written to or read from the standard I/O stream file. The parameter *op* determines the direction of the XDR stream. The value of *op* can be either XDR\_ENCODE, XDR\_DECODE, or XDR\_FREE.

The destroy routine associated with XDR streams calls fflush() on the *file* stream, but never fclose(). See fclose(3C).

A failure of any of these functions can be detected by first initializing the  $x_ops$  field in the XDR structure (*xdrs-> x\_ops*) to NULL before calling the xdr\*\_create() function. If the *x\_ops* field is still NULL, after the return from the xdr\*\_create() function, the call has failed. If the *x\_ops* field contains some other value, assume that the call has succeeded.

#### ATTRIBUTES See att

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO read(2), write(2), fclose(3C), malloc(3C), rpc(3NSL), xdr\_admin(3NSL), xdr\_complex(3NSL), xdr\_simple(3NSL), attributes(5)

# xdr\_simple(3NSL)

NAME	xdr_simple, xdr_bool, xdr_char, xdr_double, xdr_enum, xdr_float, xdr_free, xdr_hyper, xdr_int, xdr_long, xdr_longlong_t, xdr_quadruple, xdr_short, xdr_u_char, xdr_u_hyper, xdr_u_int, xdr_u_long, xdr_u_longlong_t, xdr_u_short, xdr_void – library routines for external data representation		
SYNOPSIS	<pre>#include<rpc xdr.h=""></rpc></pre>		
	<pre>bool_t xdr_bool(XDR *xdrs, bool_t *bp);</pre>		
	<pre>bool_t xdr_char(XDR *xdrs, char *cp);</pre>		
	<pre>bool_t xdr_double(XDR *xdrs, double *dp);</pre>		
	<pre>bool_t xdr_enum(XDR *xdrs, enum_t *ep);</pre>		
	<pre>bool_t xdr_float(XDR *xdrs, float *fp);</pre>		
	<pre>void xdr_free(xdrproc_t proc, char *objp);</pre>		
	<pre>bool_t xdr_hyper(XDR *xdrs, longlong_t *llp);</pre>		
	<pre>bool_t xdr_int(XDR *xdrs, int *ip);</pre>		
	<pre>bool_t xdr_long(XDR *xdrs, longt *lp);</pre>		
	<pre>bool_t xdr_longlong_t(XDR *xdrs, longlong_t *llp);</pre>		
	<pre>bool_t xdr_quadruple(XDR *xdrs, long double *pq);</pre>		
	<pre>bool_t xdr_short(XDR *xdrs, short *sp);</pre>		
	<pre>bool_t xdr_u_char(XDR *xdrs, unsigned char *ucp);</pre>		
	<pre>bool_t xdr_u_hyper(XDR *xdrs, u_longlong_t *ullp);</pre>		
	<pre>bool_t xdr_u_int(XDR *xdrs, unsigned *up);</pre>		
	<pre>bool_t xdr_u_long(XDR *xdrs, unsigned long *ulp);</pre>		
	<pre>bool_t xdr_u_longlong_t(XDR *xdrs, u_longlong_t *ullp);</pre>		
	<pre>bool_t xdr_u_short(XDR xdrs, unsigned short *usp);</pre>		
	<pre>bool_t xdr_voidvoid););</pre>		
DESCRIPTION	The XDR library routines allow C programmers to describe simple data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.		
	These routines require the creation of XDR streams (see xdr_create(3NSL)).		
Routines	See rpc(3NSL) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that malloc(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.		
	xdr_bool()xdr_bool() translates between booleans (C integers)and their external representations. When encoding		

	data, this filter produces values of either 1 or 0. This routine returns TRUE if it succeeds, FALSE otherwise.
xdr_char()	<pre>xdr_char() translates between C characters and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. Note: encoded characters are not packed, and occupy 4 bytes each. For arrays of characters, it is worthwhile to consider xdr_bytes(), xdr_opaque(), or xdr_string() (see xdr_complex(3NSL)).</pre>
xdr_double()	xdr_double() translates between C double precision numbers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
xdr_enum()	xdr_enum() translates between C enums (actually integers) and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
xdr_float()	xdr_float() translates between C floats and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
xdr_free()	Generic freeing routine. The first argument is the XDR routine for the object being freed. The second argument is a pointer to the object itself. Note: the pointer passed to this routine is not freed, but what it points to is freed (recursively, depending on the XDR routine).
xdr_hyper()	xdr_hyper() translates between ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
xdr_int()	<pre>xdr_int() translates between C integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.</pre>
xdr_long()	xdr_long() translates between C long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	In a 64-bit environment, this routine returns an error if the value of lp is outside the range [INT32_MIN, INT32_MAX]. The xdr_int() routine is recommended in place of this routine.
<pre>xdr_longlong_t()</pre>	<pre>xdr_longlong_t() translates between ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. This routine is identical to xdr_hyper().</pre>

# xdr\_simple(3NSL)

	xdr_quadruple()	xdr_quadruple() translates between IEEE quadruple precision floating point numbers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	xdr_short()	xdr_short() translates between C short integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	xdr_u_char()	xdr_u_char() translates between unsigned C characters and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	xdr_u_hyper()	xdr_u_hyper() translates between unsigned ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	<pre>xdr_u_int()</pre>	A filter primitive that translates between a C unsigned integer and its external representation. This routine returns TRUE if it succeeds, FALSE otherwise.
	xdr_u_long()	xdr_u_long() translates between C unsigned long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
		In a 64-bit environment, this routine returns an error if the value of <i>ulp</i> is outside the range [0, UINT32_MAX]. The xdr_u_int() routine is recommended in place of this routine.
	xdr_u_longlong_t()	<pre>xdr_u_longlong_t() translates between unsigned ANSI C long long integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise. This routine is identical to xdr_u_hyper().</pre>
	xdr_u_short()	xdr_u_short() translates between C unsigned short integers and their external representations. This routine returns TRUE if it succeeds, FALSE otherwise.
	xdr_void()	This routine always returns TRUE. It may be passed to RPC routines that require a function parameter, where nothing is to be done.
ATTRIBUTES	See attributes(5) for descr	iptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

692 man pages section 3: Networking Library Functions • Last Revised 27 Aug 2001

xdr\_simple(3NSL)

SEE ALSO | malloc(3C), rpc(3NSL), xdr\_admin(3NSL), xdr\_complex(3NSL), xdr\_create(3NSL), attributes(5)

xfn(	(3XFN)

n(3XFN)	
NAME	xfn – overview of the XFN interface
DESCRIPTION	The primary service provided by a federated naming system is to map a <i>composite name</i> to a <i>reference</i> . A composite name is composed of name components from one or more naming systems. A reference consists of one or more communication end points. An additional service provided by a federated naming system is to provide access to attributes associated with named objects. This extension is to satisfy most applications' additional naming service needs without cluttering the basic naming service model. XFN is a programming interface for a federated naming service.
	To use the XFN interface, include the $xfn/xfn$ . h header file and link the application with $-lxfn$ .
	The $xfn/xfn$ . h header file contains the interface declarations for:
	<ul> <li>the XFN base context interface,</li> </ul>
	<ul> <li>the XFN base attribute interface,</li> </ul>
	<ul> <li>status object and status codes used by operations in these two interfaces,</li> </ul>
	<ul> <li>abstract data types passed as parameters to and returned as values from operations in these two interfaces, and</li> </ul>
	<ul> <li>the interface for the XFN standard syntax model for parsing compound names.</li> </ul>
FILES	/usr/include/xfn/xfn.h
SEE ALSO	<pre>FN_ctx_t(3XFN), FN_status_t(3XFN), xfn_attributes(3XFN), xfn_composite_names(3XFN), xfn_compound_names(3XFN), xfn_status_codes(3XFN), fns(5), fns_policies(5)</pre>
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

#### NAME | xfn\_attributes - an overview of XFN attribute operations

#### DESCRIPTION

XFN assumes the following model for attributes. A set of zero or more attributes is associated with a named object. Each attribute in the set has a unique attribute identifier, an attribute syntax, and a (possibly empty) set of distinct data values. Each attribute value has an opaque data type. The attribute identifier serves as a name for

the attribute. The attribute syntax indicates how the value is encoded in the buffer.

The operations of the base attribute interface may be used to examine and modify the settings of attributes associated with existing named objects. These objects may be contexts or other types of objects. The attribute operations do not create names or remove names from contexts.

The range of support for attribute operations may vary widely. Some naming systems may not support any attribute operations. Other naming systems may only support read operations, or operations on attributes whose identifiers are in some fixed set. A naming system may limit attributes to have a single value, or may require at least one value. Some naming systems may only associate attributes with context objects, while others may allow associating attributes with non-context objects.

These are the interfaces:

#include <xfn/xfn.h>

```
FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name,
FN status t *status);
```

- FN\_attrvalue\_t \*fn\_valuelist\_next(FN\_valuelist\_t \*vl, FN\_identifier\_t \*\*attr\_syntax, FN status t \*status);

void fn\_valuelist\_destroy(FN\_valuelist\_t \*vl, FN\_status\_t \*status);

FN\_attribute\_t \*fn\_multigetlist\_next(FN\_multigetlist\_t \*ml,

Networking Library Functions 695

xfn\_attributes(3XFN)

o alco(o)(i) i i	)		
	<pre>FN_status_t *status);</pre>		
	void fn_multigetlist_destroy(FN_multigetlist_t *ml, FN_status_t *status)		
	<pre>int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,</pre>		
	<pre>FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx,</pre>		
	The following describes briefly the operations in the base attribute interface. Detailed descriptions are given in the respective reference manual pages for these operations.		
	<pre>fn_attr_get() returns the attribute identified. fn_attr_modify() modifies the attribute identified as described by mod_op.</pre>		
	<pre>fn_attr_get_ids() returns the identifiers of the attributes of the named object.</pre>		
	fn_attr_get_values() and its set of related operations are used for returning the individual values of an attribute.		
	<pre>fn_attr_multi_get() and its set of related operations are used for returning the requested attributes associated with the named object. fn_attr_multi_modify() modifies multiple attributes associated with the named object in a single invocation.</pre>		
	<pre>fn_ctx_get_syntax_attrs() returns the syntax attributes associated with the named context.</pre>		
ERRORS	<pre>status is set as described in FN_status_t(3XFN) and xfn_status_codes(3XFN). The following status codes are of special relevance to attribute operations:</pre>		
	FN_E_ATTR_VALUE_REQUIRED	The operation attempted to create an attribute without a value, and the specific naming system does not allow this.	
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to perform the attempted attribute operation.	
	FN_E_INSUFFICIENT_RESOURCES	There are insufficient resources to retrieve the requested attribute(s).	
	FN_E_INVALID_ATTR_IDENTIFIER	The attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier.	
	FN_E_INVALID_ATTR_VALUE	One of the values supplied was not in the appropriate form for the given attribute.	

xfn\_attributes(3XFN)

		,un_attributeb(6),u i i i
	FN_E_NO_SUCH_ATTRIBUTE	The object did not have an attribute with the given identifier.
	FN_E_TOO_MANY_ATTR_VALUES	The operation attempted to associate more values with an attribute than the naming system supported.
USAGE	<b>GE</b> Except for fn_ctx_get_syntax_attrs(), an attribute operation using a name is not necessarily equivalent to an independent fn_ctx_lookup() of followed by an attribute operation in which the caller supplies the resulting and an empty name. This is because there is a range of attribute models in w attribute is associated with a name in a context, or an attribute is associated object named, or both. XFN accommodates all of these alternatives. Invoking attribute operation using the target context and the terminal atomic name ac either the attributes that are associated with the target name or target name of this is dependent on the underlying attribute model. This document uses the <i>attributes associated with a named object</i> to refer to all of these cases.	
	XFN specifies no guarantees about the re- reference associated with a given name. So bound to a name in one or more attribute operations might affect the information u	Some naming systems may store the reference es associated with a name. Attribute
	1	en the intention is to manipulate a reference. ecific knowledge about how an XFN context
SEE ALSO	FN_attribute_t(3XFN), FN_attrset FN_composite_name_t(3XFN), FN_ct FN_status_t(3XFN), fn_attr_get(3) fn_attr_get_values(3XFN), fn_att fn_attr_multi_get(3XFN), fn_attr fn_ctx_get_syntax_attrs(3XFN), f xfn_status_codes(3XFN)	x_t(3XFN),FN_identifier_t(3XFN), (FN),fn_attr_get_ids(3XFN), r_modify(3XFN), _multi_modify(3XFN),
NOTES	specification. It is likely that there will be changes in the final version of this specifi	anges to applications that have been

Networking Library Functions 697

xfn\_composite\_names(3XFN)

NAME	xfn_composite_names – XFN composite syntax: an overview of the syntax for XFN composite name
DESCRIPTION	An <i>XFN composite name</i> consists of an ordered list of zero or more components. Each component is a string name from the namespace of a single naming system. It may be an atomic or a compound name in that namespace.
	XFN defines an abstract data type, FN_composite_name_t, for representing the structural form of a composite name. XFN also defines a standard string form for composite names. This form is the concatenation of the components of a composite name from left to right with the <i>XFN component separator</i> ('/') character to separate each component.
	These are the interfaces:
	<pre>#include <xfn xfn.h=""> FN_composite_name_t *fn_composite_name_from_string( const FN_string_t *str); FN_string_t *fn_string_from_composite_name( const FN_composite_name_t *name);</xfn></pre>
	The function fn_composite_name_from_string parses the string representation of a composite name into its corresponding composite name object FN_composite_name_t. The function fn_string_from_composite_name composes the string representation of a composite name given its composite name object form FN_composite_name_t.
APPLICATION USAGE	Special characters used in the XFN composite name syntax, such as the separator or escape characters, have the same encoding as they would in ISO 646.
	All XFN implementations are required to support the portable representation, ISO 646. All other representations are optional.
	All characters of the string form of a XFN composite name use a single encoding. This does not preclude component names of a composite name in its structural form from having different encodings. Code set mismatches that occur during the process of coverting a composite name structure to its string form are resolved in an implementation-dependent way. When an implementation discovers that a composite name has components with incompatible code sets, it returns the error code FN_E_INCOMPATIBLE_CODE_SETS.
SEE ALSO	<pre>FN_string_t(3XFN), FN_compound_name_t(3XFN), xfn(3XFN)</pre>

NAME | xfn\_compound\_names – XFN compound syntax: an overview of XFN model for compound name parsing

**DESCRIPTION** Each naming system in an XFN federation has a naming convention. XFN defines a standard model of expressing compound name syntax that covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention.

The model uses the attributes in the following table to describe properties of the syntax. Unless otherwise qualified, these syntax attributes have attribute identifiers that use the FN\_ID\_STRING format. A context that supports the XFN standard syntax model has an attribute set containing the fn\_syntax\_type (with identifier format FN\_ID\_STRING) attribute with the value "standard" (ASCII attribute syntax).

These are the interfaces:

```
#include <xfn/xfn.h>
FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx,
const FN_composite_name_t *name,
FN_status_t *status);
FN_compound_name_t *fn_compound_name_from_syntax_attrs(const FN_attrset_t *aset,
const FN_string_t *name, FN_status_t *status);
```

fn\_syntax\_type

Its value is the ASCII string "standard" if the context supports the XFN standard syntax model. Its value is an implementation-specific value if another syntax model is supported.

fn\_std\_syntax\_direction

Its value is an ASCII string, one of "left\_to\_right", "right\_to\_left", or "flat". This determines whether the order of components in a compound name string goes from left to right, right to left, or whether the namespace is flat (in other words, not hierarchical; em all names are atomic).

fn\_std\_syntax\_separator

Its value is the separator string for this name syntax. This attribute is required unless the fn\_std\_syntax\_direction is "flat".

fn\_std\_syntax\_escape

If present, its value is the escape string for this name syntax.

fn\_std\_syntax\_case\_insensitive

If this attribute is present, it indicates that names that differ only in case are considered identical. If this attribute is absent, it indicates that case is significant. If a value is present, it is ignored.

fn\_std\_syntax\_begin\_quote

If present, its value is the begin-quote string for this syntax. There can be multiple values for this attribute.

#### xfn\_compound\_names(3XFN)

fn std syntax end quote

If present, its value is the end-quote string for this syntax. There can be multiple values for this attribute.

fn\_std\_syntax\_ava\_separator

If present, its value is the attribute value assertion separator string for this syntax.

fn\_std\_syntax\_typeval\_separator
 If present, its value is the attribute type-value separator string for this syntax.

fn\_std\_syntax\_code\_sets

If present, its value identifies the code sets of the string representation for this syntax. Its value consists of a structure containing an array of code sets supported by the context; the first member of the array is the preferred code set of the context. The values for the code sets are defined in the X/Open code set registry. If this attribute is not present, or if the value is empty, the default code set is ISO 646 (same encoding as ASCII).

fn std syntax locale info

If present, identifies locale information, such as character set information, of the string representation for this syntax. The interpretation of its value is implementation-dependent.

The XFN standard syntax attributes are interpreted according to the following rules:

- 1. In a string without quotes or escapes, any instance of the separator string delimits two atomic names.
- 2. A separator, quotation or escape string is escaped if preceded immediately (on the left) by the escape string.
- 3. A non-escaped begin-quote which precedes a component must be matched by a non-escaped end-quote at the end of the component. Quotes embedded in non-quoted names are treated as simple characters and do not need to be matched. An unmatched quotation fails with the status code FN\_E\_ILLEGAL\_NAME.
- 4. If there are multiple values for begin-quote and end-quote, a specific begin-quote value must be matched with its corresponding end-quote value.
- 5. When the separator appears between a (non-escaped) begin quote and the end quote, it is ignored.
- 6. When the separator is escaped, it is ignored. An escaped begin-quote or end-quote string is not treated as a quotation mark. An escaped escape string is not treated as an escape string.
- 7. A non-escaped escape string appearing within quotes is interpreted as an escape string. This can be used to embed an end-quote within a quoted string.

After constructing a compound name from a string, the resulting component atoms have one level of escape strings and quotations interpreted and consumed.

 $\texttt{fn\_ctx\_get\_syntax\_attrs()}$  is used to obtain the syntax attributes associated with a context.

xfn\_compound\_names(3XFN)

fn compound name from syntax() is used to construct a compound name object using the string form of the name and the syntax attributes of the name. ERRORS FN E ILLEGAL NAME The name supplied to the operation was not a well-formed component according to the name syntax of the context. Code set mismatches that occur during the FN E INCOMPATIBLE CODE SETS construction of the compound name's string form are resolved in an implementation-dependent way. When an implementation discovers that a compound name has components with incompatible code sets, it returns this error code. FN E INVALID\_SYNTAX\_ATTRS The syntax attributes supplied are invalid or insufficient to fully specify the syntax. FN E SYNTAX NOT SUPPORTED The syntax specified is not supported. USAGE Most applications treat names as opaque data. Hence, the majority of clients of the

- XFN interface will not need to parse compound names from specific naming systems. Some applications, however, such as browsers, need such capabilities. These applications would use fn\_ctx\_get\_syntax\_attrs() to obtain the syntax-related attributes of a context and, if the context uses the XFN standard syntax model, it would examine these attributes to determine the name syntax of the context.
- SEE ALSO FN\_attribute\_t(3XFN), FN\_attrset\_t(3XFN), FN\_compound\_name\_t(3XFN), FN\_identifier\_t(3XFN), FN\_string\_t(3XFN) fn\_ctx\_get\_syntax\_attrs (3XFN), xfn(3XFN)
  - **NOTES** The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

xfn links(3XFN)

xfn\_links - XFN links: an overview of XFN links NAME

DESCRIPTION

An XFN link is a special form of reference that contains a composite name, the link *name*, and that may be bound to an atomic name in an XFN context. Because the link name is a composite name, it may span multiple namespaces.

Normal resolution of names in context operations always follows XFN links. If the first composite name component of the link name is the atomic name ".", the link name is resolved relative to the same context in which the link is bound, otherwise, the link name is resolved relative to the XFN Initial Context of the client. The link name may itself cause resolution to pass through other XFN links. This gives rise to the possibility of a cycle of links whose resolution could not terminate normally. As a simple means to avoid such non-terminating resolutions, implementations may define limits on the number of XFN links that may be resolved in any single operation invoked by the caller.

These are the interfaces:

#include <xfn/xfn.h>

FN ref t \*fn ref create link(const FN composite name t \*link\_name); int fn ref is link(const FN ref t \*ref); FN\_composite\_name\_t \*fn\_ref\_link\_name( const FN\_ref\_t \*link\_ref); FN ref t \*fn ctx lookup link(FN ctx t \*ctx, const FN composite name t \*name, FN\_status\_t \*status); unsigned int fn status link code(const FN status t \*stat); const FN composite name t \*fn status link remaining name( const FN status t \*stat); const FN\_composite\_name\_t \*fn\_status\_link\_resolved\_name( const FN status t \*stat); const FN ref t \*fn status link resolved ref( const FN status t \*stat); int fn status set link code(FN status t \*stat, unsigned int code); int fn status set link remaining name(FN status t \*stat, const FN composite name t \*name); int fn status set link resolved name(FN status t \*stat, const FN\_composite\_name\_t \*name); int fn status set link resolved ref(FN status t \*stat, const FN\_ref\_t \*ref);

Links are bound to names using the normal fn ctx bind() and unbound using the normal fn ctx unbind() operation. The operation fn ref create link() is provided for constructing a link reference from a composite name. Since normal

	resolution always follows links, a separate operation, fn_ctx_lookup_link() is		
	<pre>provided to lookup the link itself. In the case that an error occurred while resolving an XFN link, the status object set by the operation contains additional information about that error and sets the corresponding link status fields using fn_status_set_link_code(), fn_status_set_link_remaining_name(), fn_status_set_link_resolved_name() and fn_status_set_link_resolved_ref(). The link status fields can be retrieved using fn_status_link_code(), fn_status_link_remaining_name(), fn_status_link_resolved_name() and fn_status_link_resolved_ref().</pre>		
ERRORS	The following status codes are of special relevance when performing operations involving XFN links:		
	FN_E_LINK_ERROR	There was an error encountered resolving an XFN link encountered during resolution of the supplied name. Check the link part of the status object to determine cause of the link error.	
	FN_E_LINK_LOOP_LIMIT	A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller.	
	FN_E_MALFORMED_LINK	A malformed link reference was encountered. For the fn_ctx_lookup_link() operation, the name supplied resolved to a reference that was not a link.	
APPLICATION USAGE	For the fn_ctx_bind(), fn_ctx_unbind(), fn_ctx_rename(), fn_ctx_lookup_link(), fn_ctx_create_subcontext() and fn_ctx_destroy_subcontext() operations, resolution of the given name continues to the target context — that named by all but the terminal atomic part of the given name; the terminal atomic name is not resolved. Consequently, for operations that involve unbinding the terminal atomic part such as fn_ctx_unbind(), if the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.		
	Many naming systems support a native notion of link that may be used within the naming system itself. XFN does not determine whether there is any relationship between such native links and XFN links.		
SEE ALSO	<pre>fn_ctx_bind(3XFN), fn_ct fn_ctx_lookup(3XFN), fn_</pre>	FN),FN_ref_t(3XFN),FN_status_t(3XFN), xx_destroy_subcontext(3XFN), _ctx_lookup_link(3XFN),fn_ctx_rename(3XFN), n_status_codes(3XFN),xfn(3XFN)	

NAME	xfn_status_codes - descriptions of XFN status codes		
SYNOPSIS	<pre>#include <xfn xfn.h=""></xfn></pre>		
DESCRIPTION	The result status of operations in the context interface and the attribute interface is encapsulated in an FN_status_t object. This object contains information about how the operation completed: whether an error occurred in performing the operation; if so, what kind of error; and information localizing where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.		
	The context status object consists of several items of information. One of them is the primary status code, describing the disposition of the operation. In the case that an error occurred while resolving an XFN link, the primary status code has the value FN_E_LINK_ERROR, and the link status code describes the error that occurred while resolving the XFN link.		
XFN Status Codes	Both the primary status code and the link status code are values of type unsigned int that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. Currently, values and interpretations for the following codes are determined by XFN.		
	FN_SUCCESS	The operation succeeded.	
	FN_E_ATTR_NO_PERMISSION	The caller did not have permission to perform the attempted attribute operation.	
	FN_E_ATTR_VALUE_REQUIRED	The operation attempted to create an attribute without a value, and the specific naming system does not allow this.	
	FN_E_AUTHENTICATION_FAILURE	The identity of the client principal could not be verified.	
	FN_E_COMMUNICATION_FAILURE	An error occurred in communicating with one of the contexts involved in the operation.	
	FN_E_CONFIGURATION_ERROR	A problem was detected that indicated an error in the installation of the XFN implementation.	
	FN_E_CONTINUE	The operation should be continued using the remaining name and the resolved reference returned in the status.	
	FN_E_CTX_NO_PERMISSION	The client did not have permission to perform the operation.	
	FN_E_CTX_NOT_EMPTY	(Applies only to fn_ctx_destroy_subcontext().) The	

	naming system required that the context be empty before its destruction, and it was not empty.
FN_E_CTX_UNAVAILABLE	Service could not be obtained from one of the contexts involved in the operation. This may be because the naming system is busy, or is not providing service. In some implementations this may not be distinguished from a communication failure.
FN_E_ILLEGAL_NAME	The name supplied to the operation was not a well- formed XFN composite name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.
FN_E_E_INCOMPATIBLE_CODE_SETS	The operation involved character strings of incompatible code sets, or the supplied code set is not supported by the implementation.
FN_E_INSUFFICIENT_RESOURCES	Either the client or one of the involved contexts could not obtain sufficient resources (for example, memory, file descriptors, communication ports, stable media space, and so on) to complete the operation successfully.
FN_E_INVALID_ATTR_IDENTIFIER	The attribute identifier was not in a format acceptable to the naming system, or its content was not valid for the format specified for the identifier.
FN_E_INVALID_ATTR_VALUE	One of the values supplied was not in the appropriate form for the given attribute.
FN_E_INVALID_ENUM_HANDLE	The enumeration handle supplied was invalid, either because it was from another enumeration, or because an update operation occurred during the enumeration, or because of some other reason.
FN_E_INVALID_SYNTAX_ATTRS	The syntax attributes supplied are invalid or insufficient to fully specify the syntax.
FN_E_LINK_ERROR	There was an error in resolving an XFN link encountered during resolution of the supplied name.
FN_E_LINK_LOOP_LIMIT	A non-terminating loop (cycle) in the resolution can arise due to XFN links

Networking Library Functions 705

	encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller.
FN_E_MALFORMED_LINK	A malformed link reference was encountered. For fn_ctx_lookup_link(), the name supplied resolved to a reference that was not a link.
FN_E_MALFORMED_REFERENCE	A context object could not be constructed from the supplied reference, because the reference was not properly formed.
FN_E_NAME_IN_USE	(Only for operations that bind names.) The supplied name was already in use.
FN_E_NAME_NOT_FOUND	Resolution of the supplied composite name proceeded to a context in which the next atomic component of the name was not bound.
FN_E_NO_SUCH_ATTRIBUTE	The object did not have an attribute with the given identifier.
FN_E_NO_SUPPORTED_ADDRESS	A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.
FN_E_NOT_A_CONTEXT	Either one of the intermediate atomic names did not name a context, and resolution could not proceed beyond this point, or the operation required that the caller supply the name of a context, and the name did not resolve to a reference for a context.
FN_E_OPERATION_NOT_SUPPORTED	The operation attempted is not supported.
FN_E_PARTIAL_RESULT	The operation attempted is returning a partial result.
FN_E_SYNTAX_NOT_SUPPORTED	The syntax type specified is not supported.
FN_E_TOO_MANY_ATTR_VALUES	The operation attempted to associate more values with an attribute than the naming system supported.

	FN_E_UNSPECIFIED_ERROR	An error occurred that could not be classified by any of the other error codes.
FILES	<pre>#include <xfn xfn.h=""></xfn></pre>	XFN status codes header file
SEE ALSO	<pre>FN_status_t(3XFN), xfn(3XFN)</pre>	
NOTES	The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As	

developed against the preliminary specification.

the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been

ypclnt(3NSL)

NAME	ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err – NIS Version 2 client interface		
SYNOPSIS	<pre>cc -flag filelnsl [ library ] #include <rpcsvc ypclnt.h=""> #include <rpcsvc yp_prot.h=""></rpcsvc></rpcsvc></pre>		
	<pre>yp_bind(char *indomain);</pre>		
	<pre>void yp_unbind (char *indomain);</pre>		
	<pre>yp_get_default_domain(char **outdomain);</pre>		
	<pre>yp_match(char *indomain, char *inmap, char *inkey, int inkeylen, char</pre>		
	<pre>yp_first(char *indomain, char *inmap, char **outkey, int *outkeylen,</pre>		
	<pre>yp_next(char *indomain, char *inmap, char *inkey, int *inkeylen, char                                  *outkey, int *outkeylen, char **outval, int *outvallen);</pre>		
	<pre>yp_all(char *indomain, char *inmap, struct ypall_callback *incallback);</pre>		
	<pre>yp_order(char *indomain, char *inmap, unsigned long *outorder);</pre>		
	<pre>yp_master(char *indomain, char *inmap, char **outname);</pre>		
	<pre>char *yperr_string(int incode);</pre>		
	<pre>ypprot_err(unsigned int incode);</pre>		
DESCRIPTION	This package of functions provides an interface to NIS, Network Information Service Version 2, formerly referred to as YP. In this version of SunOS, NIS version 2 is supported only for compatibility with previous versions. The recommended enterprise level information service is NIS+ or NIS version 3. See nis+(1). The current SunOS supports only the client interface to NIS version 2. This client interface will in turn be served either by an existing ypserv process running on another machine on the network that has an earlier version of SunOS, or by a NIS+ server running in "YP-compatibility mode." See rpc.nisd(1M). The NOTES section in ypfiles(4) discusses the implications of being an NIS client of an NIS+ server in "YP-compatibility mode." For commands to access NIS from a client machine, see ypbind(1M), ypwhich(1), ypmatch(1), and ypcat(1). The package can be loaded from the standard library, /usr/lib/libnsl.so.1.		
	All input parameter names begin with <i>in</i> . Output parameters begin with <i>out</i> . Output parameters of type char ** should be addresses of uninitialized character pointers. Memory is allocated by the NIS client package using malloc(3C) and can be freed by the user code if it has no continuing need for it. For each <i>outkey</i> and <i>outval</i> , two extra bytes of memory are allocated at the end that contain NEWLINE and null, respectively, but these two bytes are not reflected in <i>outkeylen</i> or <i>outvallen</i> . The <i>indomain</i> and <i>inmap</i> strings must be non-null and null-terminated. String parameters		

that are accompanied by a count parameter may not be null, but they may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions in this package of type *int* return 0 if they succeed. Otherwise, they return a failure code (YPERR\_*xxxx*). Failure codes are described in the ERRORS section.

#### Routines | yp bind()

To use the NIS name services, the client process must be "bound" to an NIS server that serves the appropriate domain using yp\_bind(). Binding need not be done explicitly by user code. Binding is done automatically whenever an NIS lookup function is called. yp\_bind() can be called directly for processes that make use of a backup strategy, for example, a local file in cases when NIS services are not available. If a process calls yp\_bind(), it should call yp\_unbind() when it is finished using NIS in order to free up resources.

#### yp\_unbind()

Each binding allocates or uses up one client process socket descriptor. Each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. yp\_unbind() is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to yp\_unbind() makes the domain *unbound*, and frees all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the ypclnt() layer will retry a few more times or until the operation succeeds, provided that rpcbind(1M) and ypbind(1M) are running, and either:

- The client process cannot bind a server for the proper domain; or
- RPC requests to the server fail.

Under the following circumstances, the ypclnt layer will return control to the user code, with either an error or success code and the results:

- If an error is not RPC-related.
- If rpcbind is not running.
- If ypbind is not running.
- If a bound ypserv process returns any answer (success or failure).

#### yp\_get\_default\_domain ()

The NIS lookup calls require a map name and a domain name, at minimum. The client process should know the name of the map of interest. Client processes fetch the node's default domain by calling yp\_get\_default\_domain() and use the returned *outdomain* as the *indomain* parameter to successive NIS name service calls. The domain returned is the same as that returned using the SI\_SRPC\_DOMAIN command to the sysinfo(2) system call. The value returned in *outdomain* should not be freed.

#### ypclnt(3NSL)

#### yp match()

yp\_match() returns the value associated with a passed key. This key must be exact because no pattern matching is available. yp\_match() requires a full YP map name, such as hosts.byname, instead of the nickname hosts.

#### yp\_first()

yp\_first() returns the first key-value pair from the named map in the named domain.

#### yp next()

yp\_next() returns the next key-value pair in a named map. The *inkey* parameter must be the *outkey* returned from an initial call to yp\_first() (to get the second key-value pair) or the one returned from the *n*th call to yp\_next() (to get the *n*th + second key-value pair). Similarly, the *inkeylen* parameter must be the *outkeylen* returned from the earlier yp\_first() or yp\_next() call.

The concept of first and next is particular to the structure of the NIS map being processed. Retrieval order is not related to either the lexical order within any original (non-NIS name service) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee is that if the yp\_first() function is called on a particular map, and then the yp\_next() function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR\_NOMORE, every entry in the data base is seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries are seen in the same order.

Under conditions of heavy server load or server failure, the domain can become unbound, then bound once again (perhaps to a different server) while a client is running. This binding can cause a break in one of the enumeration rules. Specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. For a better solution to enumerating all entries in a map, see yp all().

#### yp\_all()

The function yp\_all() provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction takes place as a single RPC request and response. yp\_all() can be used just like any other NIS name service procedure to identify the map in the normal manner and to supply the name of a function that will be called to process each key-value pair within the map. The call to yp\_all() returns only when the transaction is completed (successfully or unsuccessfully), or the foreach() function decides that it does not want to see any more key-value pairs.

The third parameter to yp\_all() is:

```
struct ypall_callback *incallback {
    int (*foreach)();
    char *data;
};
```

The function foreach() is called:

foreach(int instatus, char \*inkey, int inkeylen, char \*inval, int invallen, char \*indata);

The *instatus* parameter holds one of the return status values defined in <rpcsvc/yp\_prot.h>, either YP\_TRUE or an error code. See ypprot\_err(), for a function that converts an NIS name service protocol error code to a ypclnt layer error code.

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the yp\_all() function, and is overwritten with the arrival of each new key-value pair. The foreach() function must do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the foreach() function look exactly as they do in the server's map. If they were not NEWLINE-terminated or null-terminated in the map, they would not be here either.

The *indata* parameter is the contents of the *incallback->data* element passed to yp\_all(). The data element of the callback structure can be used to share state information between the foreach() function and the mainline code. Its use is optional, and no part of the NIS client package inspects its contents; cast it to something useful, or ignore it. The foreach() function is Boolean. It should return 0 to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If foreach() returns a non-zero value, it is not called again. The functional value of yp\_all() is then 0.

#### yp order()

yp\_order() returns the order number for a map. This function is not supported if the ypbind process on the client's system is bound to an NIS+ server running in "YP-compatibility mode."

yp\_master()

yp\_master() returns the machine name of the master NIS server for a map.

#### yperr string()

yperr\_string() returns a pointer to an error message string that is null-terminated but contains no period or NEWLINE.

#### ypprot err()

ypprot\_err() takes an NIS name service protocol error code as input, and returns a ypclnt() layer error code, which can be used as an input to yperr string().

# **RETURN VALUES** All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails:

YPERR ACCESS Access violation.

Networking Library Functions 711

ypclnt(3NSL)

	YPERR_BADARGS	The arguments to the function are bad.
	YPERR_BADDB	The YP database is bad.
	YPERR_BUSY	The database is busy.
	YPERR_DOMAIN	Cannot bind to server on this domain.
	YPERR_KEY	No such key in map.
	YPERR_MAP	No such map in server's domain.
	YPERR_NODOM	Local domain name not set.
	YPERR_NOMORE	No more records in map database.
	YPERR_PMAP	Cannot communicate with rpcbind.
	YPERR_RESRC	Resource allocation failure.
	YPERR_RPC	RPC failure; domain has been unbound.
	YPERR_YPBIND	Cannot communicate with ypbind.
	YPERR_YPERR	Internal YP server or client error.
	YPERR_YPSERV	Cannot communicate with ypserv.
	YPERR_VERS	YP version mismatch.
FILES	/usr/lib/libnsl.so.1	

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

SEE ALSO nis+(1), ypcat(1), ypmatch(1), ypwhich(1), rpc.nisd(1M), rpcbind(1M), ypbind(1M), ypserv(1M), sysinfo(2), malloc(3C), ypfiles(4), attributes(5)

NAME	yp_update – change NIS information		
SYNOPSIS	<pre>#include <rpcsvc ypclnt.h=""></rpcsvc></pre>		
	<pre>int yp_update(char *domain, char *map, unsigned ypop, char *key, int</pre>		
DESCRIPTION	<pre>yp_update() is used to make changes to the NIS database. The syntax is the same as that of yp_match() except for the extra parameter ypop which may take on one of four values. If it is POP_CHANGE then the data associated with the key will be changed to the new value. If the key is not found in the database, then yp_update() will return YPERR_KEY. If ypop has the value YPOP_INSERT then the key-value pair will be inserted into the database. The error YPERR_KEY is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass ypop as YPOP_STORE and no error will be returned if the key already or does not exist. To delete an entry, the value of ypop should be YPOP_DELETE.</pre>		
	This routine depends upon secure RPC, and will not work unless the network is running secure RPC.		
RETURN VALUES	If the value of <i>ypop</i> is POP_CHANGE, yp_update() returns the error YPERR_KEY if the key is not found in the database.		
	If the value of <i>ypop</i> is POP_INSERT, yp_update() returns the error YPERR_KEY if the key already exists in the database.		
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:		
	MT-Level	Unsafe	
	INIT-LEVEI	Unsale	
SEE ALSO	<pre>secure_rpc(3NSL), ypclnt(3NSL), attributes(5)</pre>		
NOTES	This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.		

yp\_update(3NSL)

# Index

## Α

abandon an LDAP operation in progress ldap\_abandon, 295 accept — accept a connection on a socket, 18 accept a security context initiated by a peer application — gss\_accept\_sec\_context, 200 acquire a handle for a pre-existing credential by name — gss\_acquire\_cred, 206 add a credential-element to a credential -gss\_add\_cred, 209 add an object identifier to an object identifier set — gss add oid set member, 213 address in an XFN reference — fn\_ref\_addr\_assign, 129 fn\_ref\_addr\_copy, 129 — fn\_ref\_addr\_create, 129 - fn\_ref\_addr\_data, 129 - fn\_ref\_addr\_description, 129 - fn\_ref\_addr\_destroy, 129 — fn\_ref\_addr\_length, 129 — FN\_ref\_addr\_t, 129 — fn\_ref\_addr\_type, 129 allow application to determine maximum message size with resulting output token of a specified maximum size gss\_wrap\_size\_limit, 271 gss\_wrap, 269 attribute modifications, list of - fn\_attrmodlist\_add, 75 — fn\_attrmodlist\_assign, 75

— fn\_attrmodlist\_copy, 75

attribute modifications, list of (continued) - fn\_attrmodlist\_count, 75 - fn\_attrmodlist\_create, 75 - fn\_attrmodlist\_destroy, 75 — fn\_attrmodlist\_first, 75 - fn\_attrmodlist\_next, 75 — FN\_attrmodlist\_t, 75 attribute search options fn\_search\_control\_assign, 134 — fn\_search\_control\_copy, 134 - fn\_search\_control\_create, 134 - fn\_search\_control\_destroy, 134 fn\_search\_control\_follow\_links, 134 - fn\_search\_control\_max\_names, 134 - fn\_search\_control\_return\_attr\_ids, 134 fn\_search\_control\_return\_ref, 134 - fn\_search\_control\_scope, 134 — FN\_search\_control\_t, 134 auth\_destroy — library routines for client side remote procedure call authentication, 442 authnone\_create — library routines for client side remote procedure call authentication, 442 authsys\_create — library routines for client side remote procedure call authentication, 442 authsys\_create\_default — library routines for client side remote procedure call

authentication, 442

## В

Basic Encoding Rules library decoding functions — ber\_alloc\_t, 22 — ber\_bvdup, 22 — ber\_bvecfree, 22 — ber\_bvfree, 22 — ber\_decode, 22 — ber\_first\_element, 22 — ber\_flatten, 22 — ber\_free, 22 — ber\_get\_bitstring, 22 — ber\_get\_boolean, 22 — ber\_get\_int, 22 - ber\_get\_next, 22 — ber\_get\_null, 22 — ber\_get\_stringa, 22 — ber\_get\_stringal, 22 — ber\_get\_stringb, 22 — ber\_init, 22 — ber\_next\_element, 22 — ber\_peek\_tag, 22 — ber\_scanf, 22 — ber\_skiptag, 22 ber\_alloc — simplified Basic Encoding Rules library encoding functions, 27 ber\_alloc\_t — Basic Encoding Rules library decoding functions, 22 ber\_bvdup — Basic Encoding Rules library decoding functions, 22 ber\_bvecfree— Basic Encoding Rules library decoding functions, 22 ber\_bvfree — Basic Encoding Rules library decoding functions, 22 ber\_decode — Basic Encoding Rules library decoding functions, 22 ber\_encode — simplified Basic Encoding Rules library encoding functions, 27 ber\_first\_element — Basic Encoding Rules library decoding functions, 22 ber\_flatten — Basic Encoding Rules library decoding functions, 22 ber\_free — Basic Encoding Rules library decoding functions, 22 ber\_get\_bitstring — Basic Encoding Rules library decoding functions, 22

ber\_get\_boolean — Basic Encoding Rules library decoding functions, 22 ber\_get\_int — Basic Encoding Rules library decoding functions, 22 ber\_get\_next — Basic Encoding Rules library decoding functions, 22 ber\_get\_null — Basic Encoding Rules library decoding functions, 22 ber\_get\_stringa — Basic Encoding Rules library decoding functions, 22 ber\_get\_stringal — Basic Encoding Rules library decoding functions, 22 ber\_get\_stringb — Basic Encoding Rules library decoding functions, 22 ber\_init — Basic Encoding Rules library decoding functions, 22 ber\_next\_element — Basic Encoding Rules library decoding functions, 22 ber\_peek\_tag — Basic Encoding Rules library decoding functions, 22 ber\_printf — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_bitstring — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_boolean — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_int - simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_null — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_ostring — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_seq — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_set — simplified Basic Encoding Rules library encoding functions, 27 ber\_put\_string - simplified Basic Encoding Rules library encoding functions, 27 ber\_scanf — Basic Encoding Rules library decoding functions, 22 ber\_skiptag — Basic Encoding Rules library decoding functions, 22 ber\_start\_seq — simplified Basic Encoding Rules library encoding functions, 27 ber\_start\_set — simplified Basic Encoding Rules library encoding functions, 27 bind — bind a name to a socket, 31

716 man pages section 3: Networking Library Functions • December 2001 (Beta)

bind a reference to a name — fn\_ctx\_bind, 102
bind a reference to a name and associate attributes with named object fn\_attr\_bind, 57
bind an address to a transport endpoint t\_bind, 592
byte order, convert values between host and network — byteorder, 35

- htonl, 35
- htons, 35
- ntohl, 35
- ntohs, 35

### С

change QOP, service for session — rpc\_gss\_set\_defaults, 471 character string — fn\_string\_assign, 150 — fn\_string\_bytecount, 150 — fn\_string\_charcount, 150 — fn\_string\_code\_set, 150 — fn\_string\_compare, 150 — fn\_string\_compare, 150 — fn\_string\_compare\_substring, 150

- fn\_string\_contents, 150
- fn\_string\_copy, 150
- fn\_string\_create, 150
- fn\_string\_destroy, 150
- fn\_string\_from\_contents, 150
- fn\_string\_from\_str, 150
- fn\_string\_from\_strings, 150
- fn\_string\_from\_str\_n, 150
- fn\_string\_from\_substring, 150
- fn\_string\_is\_empty, 150
- fn\_string\_next\_substring, 150
- fn\_string\_prev\_substring, 150
- fn\_string\_str, 150
- FN\_string\_t, 150

client side remote procedure call authentication, library routines for

- auth\_destroy, 442
- authnone\_create, 442
- authsys\_create, 442
- authsys\_create\_default, 442
- rpc\_clnt\_auth, 442

- clnt\_call library routines for client side calls, 444
- clnt\_control library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_create library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_create\_timed library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_create\_vers library routines for dealing
   with creation and manipulation of CLIENT
   handles, 448
- clnt\_create\_vers\_timed library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_destroy library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_dg\_create library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_door\_create library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_freeres library routines for client side calls, 444
- clnt\_geterr library routines for client side calls, 444
- clnt\_pcreateerror library routines for dealing with creation and manipulation of CLIENT handles, 448
- clnt\_perrno library routines for client side calls, 444
- clnt\_perror library routines for client side
   calls, 444
- clnt\_raw\_create library routines for dealing with creation and manipulation of CLIENT handles, 448

clnt\_spcreateerror — library routines for dealing with creation and manipulation of CLIENT handles, 448

- clnt\_sperrno library routines for client side
   calls, 444
- clnt\_sperror library routines for client side calls, 444

clnt\_tli\_create — library routines for dealing with creation and manipulation of CLIENT handles, 448 clnt\_tp\_create — library routines for dealing with creation and manipulation of CLIENT handles, 448 clnt\_tp\_create\_timed — library routines for dealing with creation and manipulation of CLIENT handles, 448 clnt\_vc\_create — library routines for dealing with creation and manipulation of CLIENT handles, 448 close an open SLP handle — SLPClose, 547 communications accept a connection on a socket accept, 18 allocate memory for, 589 bind a name to a socket — bind, 31 socketpair, 580 socket, 575 get name of peer connected to socket getpeername, 177 get socket name — getsockname, 191 initiate a connection on a socket connect, 36 listen, 364 scatter data in order to test the network spray, 583 send a message from a socket — send, sendto, sendmsg, 519 shut down part of a full-duplex connection — shutdown, 535 compare two internal-form names gss\_compare\_name, 216 component names spanning multiple naming systems — fn\_composite\_name\_append\_comp, 93 — fn\_composite\_name\_append\_name, 93 - fn\_composite\_name\_assign, 93 fn\_composite\_name\_copy, 93 — fn\_composite\_name\_count, 93 — fn\_composite\_name\_create, 93 — fn\_composite\_name\_delete\_comp, 93 — fn\_composite\_name\_destroy, 93

component names spanning multiple naming systems (continued) fn\_composite\_name\_first, 93 — fn\_composite\_name\_from\_str, 93 fn\_composite\_name\_from\_string, 93 — fn\_composite\_name\_insert\_comp, 93 — fn\_composite\_name\_insert\_name, 93 — fn\_composite\_name\_is\_empty, 93 — fn\_composite\_name\_is\_equal, 93 fn\_composite\_name\_is\_prefix, 93 — fn\_composite\_name\_is\_suffix, 93 fn\_composite\_name\_last, 93 — fn\_composite\_name\_next, 93 fn\_composite\_name\_prefix, 93 — fn\_composite\_name\_prepend\_comp, - fn\_composite\_name\_prepend\_name, 93 fn\_composite\_name\_prev, 93 — fn\_composite\_name\_suffix, 93 - FN\_composite\_name\_t, 93 — fn\_string\_from\_composite\_name, 93 configuration script execute — doconfig, 45 connect — initiate a connection on socket, 36 construct a handle to a context object using the given reference fn\_ctx\_handle\_from\_ref, 114 construct equivalent name in same context fn\_ctx\_equivalent\_name, 106 convert a contiguous string name to GSS\_API internal format — gss\_import\_name, 231 convert a GSS-API status code to text gss\_display\_status, 223 convert a mechanism name to export form gss\_export\_name, 226 convert a string to an OID gss\_str\_to\_oid, 262 convert an internal name to a mechanism name gss\_canonicalize\_name, 214 convert an OID to a string gss\_oid\_to\_str, 253 convert internal-form name to text gss\_display\_name, 221 create a copy of an internal name gss\_duplicate\_name, 225 create a security context using the RPCSEC\_GSS protocol — rpc\_gss\_seccreate, 467

718 man pages section 3: Networking Library Functions • December 2001 (Beta)

create an object-identifier set containing no object identifiers gss\_create\_empty\_oid\_set, 218 create subcontext and associate attributes fn\_attr\_create\_subcontext, 58

### D

delete a GSS-API security context gss\_delete\_sec\_context, 219 delete attributes — SLPDelAttrs, 548 deregister the SLP advertisement ---SLPDereg, 550 descriptions of XFN status codes xfn\_status\_codes, 704 determine available security mechanisms gss\_indicate\_mechs, 235 determine how long a context will remain valid — gss\_context\_time, 217 dial - establish an outgoing terminal line connection, 43 discard a credential handle ---gss\_release\_cred, 258 discard an internal-form name ---gss\_release\_name, 259 dn\_comp — resolver routines, 423 dn\_expand — resolver routines, 423 doconfig — execute a configuration script, 45

# E

endservent — get service entry, 187 escapes SLP reserved characters — SLPEscape, 552 Ethernet address mapping operations — ethers, 55 ethers — Ethernet address mapping operations, 55 external data representation See XDR, 681

## F

filter expression for attribute search - fn\_search\_filter\_arguments, 137 - fn\_search\_filter\_assign, 137 - fn\_search\_filter\_copy, 137 - fn\_search\_filter\_create, 137 - fn\_search\_filter\_destroy, 137 fn\_search\_filter\_expression, 137 — FN\_search\_filter\_t, 137 find service types — SLPFindSrvTypes, 560 fn\_attr\_bind — bind a reference to a name and associate attributes with named object, 57 fn\_attr\_create\_subcontext — create subcontext and associate attributes, 58 fn\_attr\_ext\_search — search for names whose attributes satisfy filter, 59 fn\_attr\_get — return specified attribute associated with name, 66 fn\_attr\_get\_ids — get list of attribute identifiers, 67 fn\_attr\_get\_values — return values of an attribute, 68 fn\_attribute\_add — an XFN attribute, 71 fn\_attribute\_assign — an XFN attribute, 71 fn\_attribute\_copy — an XFN attribute, 71 fn\_attribute\_create — an XFN attribute, 71 fn\_attribute\_destroy — an XFN attribute, 71 fn\_attribute\_first — an XFN attribute, 71 fn\_attribute\_identifier — an XFN attribute, 71 fn\_attribute\_next — an XFN attribute, 71 fn\_attribute\_remove — an XFN attribute, 71 fn\_attribute\_syntax — an XFN attribute, 71 FN\_attribute\_t — an XFN attribute, 71 fn\_attribute\_valuecount — an XFN attribute, 71 fn\_attr\_modify — modify specified attribute associated with name, 73 fn\_attrmodlist\_add — a list of attribute modifications, 75 fn\_attrmodlist\_assign — a list of attribute modifications, 75 fn\_attrmodlist\_copy — a list of attribute modifications, 75 fn\_attrmodlist\_count — a list of attribute modifications, 75 fn\_attrmodlist\_create — a list of attribute modifications, 75

fn\_attrmodlist\_destroy — a list of attribute modifications, 75

- fn\_attrmodlist\_first a list of attribute modifications, 75
- fn\_attrmodlist\_next a list of attribute modifications, 75
- FN\_attrmodlist\_t a list of attribute modifications, 75
- fn\_attr\_multi\_get return multiple attributes associated with named object, 78
- fn\_attr\_multi\_modify modify multiple attributes associated with named object, 82

fn\_attr\_search — search for atomic name with specified attributes in single context, 84

- fn\_attrset\_add a set of XFN attributes, 89
- fn\_attrset\_assign a set of XFN attributes, 89
- fn\_attrset\_copy a set of XFN attributes, 89
- fn\_attrset\_count a set of XFN attributes, 89 fn\_attrset\_create — a set of XFN attributes, 89
- fn\_attrset\_destroy a set of XFN
- attributes, 89
- fn\_attrset\_first a set of XFN attributes, 89
- fn\_attrset\_get a set of XFN attributes, 89
- fn\_attrset\_next a set of XFN attributes, 89
- fn\_attrset\_remove a set of XFN attributes, 89
- FN\_attrset\_t a set of XFN attributes, 89
- fn\_bindinglist\_destroy list the atomic names and references bound in a context, 116
- fn\_bindinglist\_next list the atomic names and references bound in a context, 116
- FN\_bindinglist\_t list the atomic names and references bound in a context, 116
- fn\_composite\_name\_append\_comp —
   component names spanning multiple naming
   systems, 93
- fn\_composite\_name\_append\_name —
   component names spanning multiple naming
   systems, 93
- fn\_composite\_name\_assign component names spanning multiple naming systems, 93
- fn\_composite\_name\_copy component names
   spanning multiple naming systems, 93

### fn\_composite\_name\_count — component names spanning multiple naming systems, 93

- fn\_composite\_name\_create component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_delete\_comp component names spanning multiple naming systems, 93
- fn\_composite\_name\_destroy component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_first component names spanning multiple naming systems, 93
- fn\_composite\_name\_from\_str component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_from\_string component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_insert\_comp component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_insert\_name component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_is\_empty component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_is\_equal component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_is\_prefix component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_is\_suffix component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_last component names spanning multiple naming systems, 93
- fn\_composite\_name\_next component names
   spanning multiple naming systems, 93
- fn\_composite\_name\_prefix component
   names spanning multiple naming
   systems, 93
- fn\_composite\_name\_prepend\_comp —
   component names spanning multiple naming
   systems, 93

720 man pages section 3: Networking Library Functions • December 2001 (Beta)

fn\_composite\_name\_prepend\_name component names spanning multiple naming systems, 93 fn\_composite\_name\_prev — component names spanning multiple naming systems, 93 fn\_composite\_name\_suffix — component names spanning multiple naming systems, 93 FN\_composite\_name\_t — component names spanning multiple naming systems, 93 fn\_compound\_name\_append\_comp — an XFN compound name, 98 fn\_compound\_name\_assign — an XFN compound name, 98 fn\_compound\_name\_copy — an XFN compound name, 98 fn\_compound\_name\_count — an XFN compound name, 98 fn\_compound\_name\_delete\_all — an XFN compound name, 98 fn\_compound\_name\_delete\_comp — an XFN compound name, 98 fn\_compound\_name\_destroy — an XFN compound name, 98 fn\_compound\_name\_first — an XFN compound name, 98 fn\_compound\_name\_from\_syntax\_attrs — an XFN compound name, 98 fn\_compound\_name\_get\_syntax\_attrs — an XFN compound name, 98 fn\_compound\_name\_insert\_comp — an XFN compound name, 98 fn\_compound\_name\_is\_empty — an XFN compound name, 98 fn\_compound\_name\_is\_equal — an XFN compound name, 98 fn\_compound\_name\_is\_prefix — an XFN compound name, 98 fn\_compound\_name\_is\_suffix — an XFN compound name, 98 fn\_compound\_name\_last — an XFN compound name, 98 fn\_compound\_name\_next — an XFN compound name, 98 fn\_compound\_name\_prefix — an XFN compound name, 98 fn\_compound\_name\_prepend\_comp — an XFN

compound name, 98

fn\_compound\_name\_prev — an XFN compound name, 98 fn\_compound\_name\_suffix — an XFN compound name, 98 FN\_compound\_name\_t — an XFN compound name, 98 fn\_ctx\_bind — bind a reference to a name, 102 fn\_ctx\_equivalent\_name — construct equivalent name in same context, 106 fn\_ctx\_handle\_from\_initial — return a handle to the Initial Context, 112 fn\_ctx\_handle\_from\_ref — construct a handle to a context object using the given reference, 114 fn\_ctx\_list\_bindings — list the atomic names and references bound in a context, 116 fn\_ctx\_list\_names — list the atomic names bound in a context, 117 fn\_ctx\_lookup\_link — look up the link reference bound to a name, 121 fn\_ctx\_rename — rename the name of a binding, 122 FN\_ctx\_t — an XFN context, 125 fn\_ext\_searchlist\_destroy — search for names whose attributes satisfy filter, 59 fn\_ext\_searchlist\_next — search for names whose attributes satisfy filter, 59 FN\_ext\_searchlist\_t — search for names whose attributes satisfy filter, 59 FN\_identifier\_t — an XFN identifier, 128 fn\_multigetlist\_destroy — return multiple attributes associated with named object, 78 fn\_multigetlist\_next — return multiple attributes associated with named object, 78 FN\_multigetlist\_t — return multiple attributes associated with named object, 78 fn\_namelist\_destroy — list the atomic names bound in a context, 117 fn\_namelist\_next — list the atomic names bound in a context, 117 FN\_namelist\_t — list the atomic names bound in a context, 117 fn\_ref\_addr\_assign — an address in an XFN reference, 129 fn\_ref\_addr\_copy — an address in an XFN reference, 129 fn\_ref\_addrcount — an XFN reference, 131

fn\_ref\_addr\_create — an address in an XFN reference, 129 fn\_ref\_addr\_data — an address in an XFN reference, 129 fn\_ref\_addr\_description — an address in an XFN reference, 129 fn\_ref\_addr\_destroy — an address in an XFN reference, 129 fn\_ref\_addr\_length — an address in an XFN reference, 129 FN\_ref\_addr\_t — an address in an XFN reference, 129 fn\_ref\_addr\_type — an address in an XFN reference, 129 fn\_ref\_append\_addr — an XFN reference, 131 fn\_ref\_assign — an XFN reference, 131 fn\_ref\_copy — an XFN reference, 131 fn\_ref\_create — an XFN reference, 131 fn\_ref\_create\_link — an XFN reference, 131 fn\_ref\_delete\_addr — an XFN reference, 131 fn\_ref\_delete\_all — an XFN reference, 131 fn\_ref\_description — an XFN reference, 131 fn\_ref\_destroy — an XFN reference, 131 fn\_ref\_first — an XFN reference, 131 fn\_ref\_insert\_addr — an XFN reference, 131 fn\_ref\_is\_link — an XFN reference, 131 fn\_ref\_link\_name — an XFN reference, 131 fn\_ref\_next — an XFN reference, 131 fn\_ref\_prepend\_addr — an XFN reference, 131 FN\_ref\_t — an XFN reference, 131 fn\_ref\_type — an XFN reference, 131 FNS component names spanning multiple naming systems See FN\_composite\_name\_t fn attr bind — bind a reference to a name and associate attributes with named object, 57 fn\_attr\_create\_subcontext — create subcontext and associate attributes, 58 fn\_attr\_ext\_search — search for names whose attributes satisfy filter, 59 fn\_attr\_search — search for atomic name with specified attributes in single context, 84

fn\_ctx\_equivalent\_name — construct equivalent name in same context, 106 FNS (continued) fn\_ext\_searchlist\_destroy — search for names whose attributes satisfy filter, 59 fn\_ext\_searchlist\_next — search for names whose attributes satisfy filter, 59 FN\_ext\_searchlist\_t — search for names whose attributes satisfy filter, 59 FN\_search\_control\_t — options for attribute search, 134 FN\_search\_filter\_t — filter expression for attribute search, 137 fn\_searchlist\_destroy — terminate search for atomic name with specified attributes in single context, 84 fn\_searchlist\_next — search for next atomic name with specified attributes in single context, 84 FN\_searchlist\_t — search for atomic name with specified attributes in single context, 84 fn search control assign — options for attribute search, 134 fn\_search\_control\_copy — options for attribute search, 134 fn\_search\_control\_create — options for attribute search, 134 fn search control destroy - options for attribute search, 134 fn\_search\_control\_follow\_links - options for attribute search, 134 fn\_search\_control\_max\_names - options for attribute search, 134 fn\_search\_control\_return\_attr\_ids — options for attribute search, 134 fn\_search\_control\_return\_ref — options for attribute search, 134 fn\_search\_control\_scope — options for attribute search, 134 FN\_search\_control\_t — options for attribute search, 134 fn\_search\_filter\_arguments — filter expression for attribute search, 137 fn\_search\_filter\_assign — filter expression for attribute search, 137 fn\_search\_filter\_copy — filter expression for

attribute search, 137

fn\_search\_filter\_create — filter expression for attribute search, 137 fn\_search\_filter\_destroy — filter expression for attribute search, 137 fn\_search\_filter\_expression — filter expression for attribute search, 137 FN\_search\_filter\_t — filter expression for attribute search, 137 BNF of Filter Expression, 138 Extended Operations, 140 Precedence, 138 Relational Operators, 139 Specification of Filter Expression, 138 Wildcarded Strings, 139 fn\_searchlist\_destroy — terminate search for atomic name with specified attributes in single context, 84 fn\_searchlist\_next — search for next atomic name with specified attributes in single context, 84 FN\_searchlist\_t — search for atomic name with specified attributes in single context, 84 fn\_status\_advance\_by\_name — an XFN status object, 145 fn\_status\_append\_remaining\_name — an XFN status object, 145 fn\_status\_append\_resolved\_name — an XFN status object, 145 fn\_status\_assign — an XFN status object, 145 fn\_status\_code — an XFN status object, 145 fn\_status\_copy — an XFN status object, 145 fn\_status\_create — an XFN status object, 145 fn\_status\_description — an XFN status object, 145 fn\_status\_destroy — an XFN status object, 145 fn\_status\_diagnostic\_message — an XFN status object, 145 fn\_status\_is\_success — an XFN status object, 145 fn\_status\_link\_code — an XFN status object, 145 fn\_status\_link\_diagnostic\_message — an XFN status object, 145 fn\_status\_link\_remaining\_name — an XFN status object, 145 fn\_status\_link\_resolved\_name — an XFN status object, 145

fn\_status\_link\_resolved\_ref — an XFN status object, 145 fn\_status\_remaining\_name — an XFN status object, 145 fn\_status\_resolved\_name — an XFN status object, 145 fn\_status\_resolved\_ref — an XFN status object, 145 fn\_status\_set — an XFN status object, 145 fn\_status\_set\_code — an XFN status object, 145 fn\_status\_set\_diagnostic\_message — an XFN status object, 145 fn\_status\_set\_link\_code — an XFN status object, 145 fn\_status\_set\_link\_diagnostic\_message — an XFN status object, 145 fn\_status\_set\_link\_remaining\_name — an XFN status object, 145 fn\_status\_set\_link\_resolved\_name — an XFN status object, 145 fn\_status\_set\_link\_resolved\_ref — an XFN status object, 145 fn\_status\_set\_remaining\_name — an XFN status object, 145 fn\_status\_set\_resolved\_name — an XFN status object, 145 fn\_status\_set\_resolved\_ref — an XFN status object, 145 fn\_status\_set\_success — an XFN status object, 145 FN\_status\_t — an XFN status object, 145 fn\_string\_assign — a character string, 150 fn\_string\_bytecount — a character string, 150 fn\_string\_charcount — a character string, 150 fn\_string\_code\_set — a character string, 150 fn\_string\_compare — a character string, 150 fn\_string\_compare\_substring — a character string, 150 fn\_string\_contents — a character string, 150 fn\_string\_copy — a character string, 150 fn\_string\_create — a character string, 150 fn\_string\_destroy — a character string, 150 fn string from composite name - component

names spanning multiple naming

systems, 93

fn string from compound name — an XFN compound name, 98 fn\_string\_from\_contents — a character string, 150 fn\_string\_from\_str — a character string, 150 fn\_string\_from\_strings — a character string, 150 fn\_string\_from\_str\_n — a character string, 150 fn\_string\_from\_substring — a character string, 150 fn\_string\_is\_empty — a character string, 150 fn\_string\_next\_substring — a character string, 150 fn\_string\_prev\_substring — a character string, 150 fn\_string\_str — a character string, 150 FN\_string\_t — a character string, 150 fn\_valuelist\_destroy — return values of an attribute, 68 fn\_valuelist\_next — return values of an attribute, 68 FN\_valuelist\_t — return values of an attribute, 68 fp\_resstat — resolver routines, 423 free a BerElement structure from memory ldap\_ber\_free, 298 free buffer storage allocated by a GSS-API function — gss\_release\_buffer, 257 free storage associated with a GSS-API-generated gss\_OID\_set object gss\_release\_oid\_set, 261 freeaddrinfo - translate between node name and address, 153 freehostent — get IP node entry, 164 frees memory — SLPFree, 562 functions to map Internet Protocol network interface names and interface indexes if\_nametoindex, 276

## G

gai\_strerror — translate between node name and address, 153
generic transport name-to-address translation — netdir, 367
— netdir\_free, 367 generic transport name-to-address translation (continued) — netdir\_getbyaddr, 367 - netdir\_getbyname, 367 - netdir\_mergeaddr, 367 — netdir\_options, 367 — netdir\_perror, 367 — netdir\_sperror, 367 - taddr2uaddr, 367 — uaddr2taddr, 367 get IP node entry — getipnodebyname, 164, 187 get service entry - getservbyname endservent, 187 getservbyname\_r, 187 getservbyport, 187 getservbyport\_r, 187 getservent, 187 getservent\_r, 187 setservent, 187 get credentials of client rpc\_gss\_getcred, 457 get error codes on failure — rpc\_gss\_get\_error, 459 get list of attribute identifiers fn\_attr\_get\_ids, 67 get maximum data length for transmission — rpc\_gss\_max\_data\_length, 464 — rpc\_gss\_svc\_max\_data\_length, 464 get principal names at server — rpc\_get\_principal\_name, 462 getaddrinfo - translate between node name and address, 153 getipnodebyaddr — get IP node entry, 164 getipnodebyname — get IP node entry, 164 getnameinfo - translate between node name and address, 153 getpeername — get name of peer connected to socket, 177 getpublickey — retrieve public or secret key, 183 getsecretkey - retrieve public or secret key, 183 getservbyname — get service entry, 187 getservbyname\_r — get service entry, 187 getservbyport — get service entry, 187 getservbyport\_r — get service entry, 187

getservent — get service entry, 187 getservent\_r - get service entry, 187 gss\_accept\_sec\_context — accept a security context initiated by a peer application, 200 gss\_acquire\_cred — acquire a handle for a pre-existing credential by name, 206 gss\_add\_cred — add a credential-element to a credential, 209 gss\_add\_oid\_set\_member — add an object identifier to an object identifier set, 213 gss\_canonicalize\_name — convert an internal name to a mechanism name, 214 gss\_compare\_name — compare two internal-form names, 216 gss\_context\_time — determine how long a context will remain valid, 217 gss\_create\_empty\_oid\_set — create an object-identifier set containing no object identifiers, 218 gss\_delete\_sec\_context — delete a GSS-API security context, 219 gss\_display\_name — convert internal-form name to text, 221 gss\_display\_status — convert a GSS-API status code to text, 223 gss\_duplicate\_name — create a copy of an internal name, 225 gss\_export\_name — convert a mechanism name to export form, 226 gss\_export\_sec\_context — transfer a security context to another process, 227 gss\_import\_name — convert a contiguous string name to GSS\_API internal format, 231 gss\_import\_sec\_context — import security context established by another process, 233 gss\_indicate\_mechs — determine available security mechanisms, 235 gss\_init\_sec\_context — initiate a GSS-API security context with a peer application, 236 gss\_inquire\_context — obtain information about a security context, 243 gss\_inquire\_cred — obtain information about a credential, 246

gss\_inquire\_cred\_by\_mech — obtain per-mechanism information about a credential, 248 gss\_inquire\_mechs\_for\_name - list mechanisms that support the specified name-type, 250 gss\_inquire\_names\_for\_mech — list the name-types supported by the specified mechanism, 252 gss\_oid\_to\_str — convert an OID to a string, 253 gss\_process\_context\_token — pass asynchronous token to security service, 255 gss\_release\_buffer — free buffer storage allocated by a GSS-API function, 257 gss\_release\_cred — discard a credential handle, 258 gss\_release\_name — discard an internal-form name, 259 gss\_release\_oid — release an object identifier, 260 gss\_release\_oid\_set — free storage associated with a GSS-API-generated gss\_OID\_set object, 261 gss\_str\_to\_oid — convert a string to an OID, 262 gss\_test\_oid\_set\_member — interrogate an object identifier set, 264 gss\_verify\_mic -- verify integrity of a received message, 267 gss\_wrap — verify a message with attached cryptographic message, 265, 269 gss\_wrap\_size\_limit — allow application to determine maximum message size with resulting output token of a specified maximum size, 271

#### Η

host machines, remote return information about users — rusers, rnusers, 513 hostalias — resolver routines, 423 hstrerror — resolver routines, 423 I

if\_freenameindex — functions to map Internet Protocol network interface names and interface indexes, 276 if\_indextoname — functions to map Internet Protocol network interface names and interface indexes, 276 if\_nameindex — functions to map Internet Protocol network interface names and interface indexes, 276 if\_nametoindex — functions to map Internet Protocol network interface names and interface indexes, 276 import security context established by another process — gss\_import\_sec\_context, 233 inet — Internet address manipulation, 278 inet6 — Internet address manipulation, 278 inet\_addr — Internet address manipulation, 278 inet\_lnaof — Internet address manipulation, 278 inet\_makeaddr — Internet address manipulation, 278 inet\_netof — Internet address manipulation, 278 inet\_network — Internet address manipulation, 278 inet\_ntoa — Internet address manipulation, 278 inet\_ntop — Internet address manipulation, 278 inet\_pton — Internet address manipulation, 278 initialize the LDAP library and open a connection to an LDAP server — ldap\_init, 347 — ldap\_open, 347 initiate a GSS-API security context with a peer application — gss\_init\_sec\_context, 236 Internet address manipulation — inet, 278 interrogate an object identifier set gss\_test\_oid\_set\_member, 264

# L

LDAP bind functions — ldap\_bind, 299, 324, 331, 340 Idap — Lightweight Directory Access Protocol package Index, 286 LDAP attribute remapping functions — ldap\_free\_friendlymap, 327 — ldap\_friendly\_name, 327 LDAP attribute value handling functions — ldap\_count\_values, 338 — ldap\_get\_values, 338 — ldap\_get\_values\_len, 338 LDAP character set translation functions -- ldap\_8859\_to\_t61, 302 — ldap\_enable\_translation, 302 — ldap\_set\_string\_translators, 302 -- ldap\_t61\_to\_8859, 302 — ldap\_translate\_from\_t61, 302 — ldap\_translate\_to\_t61, 302 LDAP compare operation — ldap\_compare, 304 — ldap\_compare\_ext, 304 — ldap\_compare\_ext\_s, 304 ldap\_compare\_s, 304 LDAP control disposal — ldap\_control\_free, 306 — ldap\_controls\_free, 306 LDAP delete operation — ldap\_delete, 307 — ldap\_delete\_ext, 307 — ldap\_delete\_ext\_s, 307 — ldap\_delete\_s, 307 LDAP display template functions – ldap\_disptmpl, 309 — ldap\_first\_disptmpl, 309 - ldap\_first\_tmplcol, 309 - ldap\_first\_tmplrow, 309 — ldap\_free\_templates, 309 - ldap\_init\_templates, 309 — ldap\_init\_templates\_buf, 309 — ldap\_next\_disptmpl, 309 — ldap\_next\_tmplcol, 309 — ldap\_next\_tmplrow, 309 — ldap\_oc2template, 309 - ldap\_tmplattrs, 309

LDAP DN handling functions — ldap\_dn2ufn, 328 — ldap\_dns\_to\_dn, 328 — ldap\_explode\_dn, 328 — ldap\_explode\_dns, 328 — ldap\_get\_dn, 328 — ldap\_is\_dns\_dn, 328 LDAP entry display functions — Idap\_entry2text, 315 — ldap\_entry2text\_search, 315 — ldap\_vals2text, 315 LDAP entry modification functions — ldap\_modify, 343 — ldap\_modify\_ext, 343 — ldap\_modify\_ext\_s, 343 — ldap\_modify\_s, 343 LDAP entry sorting functions — ldap\_sort, 356 — ldap\_sort\_entries, 356 — ldap\_sort\_strcasecmp, 356 — ldap\_sort\_values, 356 LDAP message processing functions — ldap\_count\_message, 326 — ldap\_first\_message, 326 — ldap\_msgtype, 326 — ldap\_next\_message, 326 LDAP message result parser - ldap\_parse\_extended\_result, 349 — ldap\_parse\_result, 349 — ldap\_parse\_sasl\_bind\_result, 349 LDAP protocol error handling functions, 318 — ldap\_err2string, 318 — ldap\_errlist, 318 — ldap\_error, 318 — ldap\_perror, 318 — ldap\_result2error, 318 LDAP search operations — ldap\_search, 352 — ldap\_search\_ext, 352 - ldap\_search\_ext\_s, 352 — ldap\_search\_s, 352 — ldap\_search\_st, 352 LDAP search preference configuration routeines — ldap\_first\_searchobj, 354 — ldap\_free\_searchprefs, 354 — ldap\_init\_searchprefs, 354

- ldap\_init\_searchprefs\_buf, 354

LDAP search preference configuration routeines (continued) – ldap\_next\_searchobj, 354 — ldap\_searchprefs, 354 LDAP Uniform Resource Locator functions — ldap\_dns\_to\_url, 360 — ldap\_dn\_to\_url, 360 — ldap\_free\_urldesc, 360 — ldap\_is\_ldap\_url, 360 - ldap\_url, 360 — ldap\_url\_parse, 360 — ldap\_url\_search, 360 — ldap\_url\_search\_s, 360 — ldap\_url\_search\_st, 360 LDAP user friendly search functions — ldap\_ufn, 358 — ldap\_ufn\_search\_c, 358 — ldap\_ufn\_search\_ct, 358 — ldap\_ufn\_search\_s, 358 — ldap\_ufn\_setfilter, 358 — ldap\_ufn\_setprefix, 358 — ldap\_ufn\_timeout, 358 ldap\_8859\_to\_t61 — LDAP character set translation functions, 302 ldap\_abandon — abandon an LDAP operation in progress, 295 ldap\_add — perform an LDAP add operation, 296 ldap\_add\_ext — perform an LDAP add operation, 296 ldap\_add\_ext\_s — perform an LDAP add operation, 296 ldap\_add\_s — perform an LDAP add operation, 296 ldap\_ber\_free — free a BerElement structure from memory, 298 ldap\_bind — LDAP bind functions, 299 General Authentication, 299 Rebinding While Following Referral, 300 Simple Authentication, 299 Unbinding, 300 ldap\_bind\_s — LDAP bind functions, 299 ldap\_build\_filter — LDAP filter generating functions, 331 ldap\_compare — LDAP compare operation, 304

ldap\_compare\_ext — LDAP compare operation, 304 ldap\_compare\_ext\_s — LDAP compare operation, 304 ldap\_compare\_s — LDAP compare operation, 304 ldap\_control\_free — LDAP control disposal, 306 ldap\_controls\_free — LDAP control disposal, 306 ldap\_count\_entries - LDAP entry parsing and counting functions, 324 ldap\_count\_messages — LDAP message processing functions, 326 ldap\_count\_references — LDAP entry parsing and counting functions, 324 ldap\_count\_values — LDAP attribute value handling functions, 338 ldap\_delete — LDAP delete operation, 307 ldap\_delete\_ext — LDAP delete operation, 307 ldap\_delete\_ext\_s — LDAP delete operation, 307 ldap\_delete\_s — LDAP delete operation, 307 ldap\_disptmpl — LDAP display template functions, 309 DISPTMPL Structure Elements, 311 Syntax IDs, 312 TMPLITEM Structure Elements, 312 ldap\_dn2ufn — LDAP DN handling functions, 328 ldap\_dns\_to\_dn — LDAP DN handling functions, 328 ldap\_dns\_to\_url — LDAP Uniform Resource Locator functions, 360 ldap\_dn\_to\_url — LDAP Uniform Resource Locator functions, 360 ldap\_enable\_translation — LDAP character set translation functions, 302 ldap\_entry2text — LDAP entry display functions, 315 ldap\_entry2text\_search — LDAP entry display functions, 315 ldap\_err2string — LDAP protocol error handling functions, 318 ldap\_errlist — LDAP protocol error handling functions, 318

ldap\_error — LDAP protocol error handling functions, 318 ldap\_explode\_dn — LDAP DN handling functions, 328 ldap\_explode\_dns — LDAP DN handling functions, 328 ldap\_first\_attribute — step through LDAP entry attributes, 322 ldap\_first\_disptmpl — LDAP display template functions, 309 ldap\_first\_entry — LDAP entry parsing and counting functions, 324 ldap\_first\_message — LDAP message processing functions, 326 ldap\_first\_reference -- LDAP entry parsing and counting functions, 324 ldap\_first\_searchobj — LDAP search preference configuration routeines, 354 ldap\_first\_tmplcol — LDAP display template functions, 309 ldap\_first\_tmplrow — LDAP display template functions, 309 ldap\_free\_friendlymap — LDAP attribute remapping functions, 327 ldap\_free\_searchprefs — LDAP search preference configuration routeines, 354 ldap\_free\_templates — LDAP display template functions, 309 ldap\_free\_urldesc — LDAP Uniform Resource Locator functions, 360 ldap\_friendly\_name — LDAP attribute remapping functions, 327 ldap\_get\_dn — LDAP DN handling functions, 328 ldap\_getfilter — LDAP filter generating functions, 331 ldap\_getfilter\_free — LDAP filter generating functions, 331 ldap\_getfirstfilter — LDAP filter generating functions, 331 ldap\_get\_lang\_values — return an attribute's values that matches a specified language subtype, 333 ldap\_get\_lang\_values\_len — return an attribute's values that matches a specified language subtype, 333

ldap\_getnextfilter — LDAP filter generating functions, 331 ldap\_get\_values — LDAP attribute value handling functions, 338 ldap\_get\_values\_len — LDAP attribute value handling functions, 338 ldap\_init — initialize the LDAP library and open a connection to an LDAP server, 347 ldap\_init\_getfilter — LDAP filter generating functions, 331 ldap\_init\_getfilter\_buf — LDAP filter generating functions, 331 ldap\_init\_searchprefs — LDAP search preference configuration routeines, 354 ldap\_init\_searchprefs\_buf — LDAP search preference configuration routeines, 354 ldap\_init\_templates — LDAP display template functions, 309 ldap\_init\_templates\_buf — LDAP display template functions, 309 ldap is dns dn — LDAP DN handling functions, 328 ldap\_is\_ldap\_url — LDAP Uniform Resource Locator functions, 360 ldap\_memcache — LDAP client caching functions, 340 ldap\_memcache\_destroy — LDAP client caching functions, 340 ldap\_memcache\_flush — LDAP client caching functions, 340 ldap\_memcache\_get — LDAP client caching functions, 340 ldap\_memcache\_init — LDAP client caching functions, 340 ldap\_memcache\_set — LDAP client caching functions, 340 ldap\_memcache\_update — LDAP client caching functions, 340 ldap\_modify — LDAP entry modification functions, 335, 343 ldap\_modify\_ext — LDAP entry modification functions, 343 ldap\_modify\_ext\_s — LDAP entry modification functions, 343 ldap\_modify\_s — LDAP entry modification functions, 343

ldap\_modrdn — modify LDAP entry RDN, 345 ldap\_modrdn2 — modify LDAP entry RDN, 345 ldap\_modrdn2\_s — modify LDAP entry RDN, 345 ldap\_modrdn\_s — modify LDAP entry RDN, 345 ldap\_msgfree — wait for and return LDAP operation result, 350 ldap\_msgtype — LDAP message processing functions, 326 ldap\_next\_attribute — step through LDAP entry attributes, 322 ldap\_next\_disptmpl — LDAP display template functions, 309 ldap\_next\_entry — LDAP entry parsing and counting functions, 324 ldap\_next\_message — LDAP message processing functions, 326 ldap\_next\_reference — LDAP entry parsing and counting functions, 324 ldap\_next\_searchobj — LDAP search preference configuration routeines, 354 ldap\_next\_tmplcol — LDAP display template functions, 309 ldap\_next\_tmplrow — LDAP display template functions, 309 ldap\_oc2template — LDAP display template functions, 309 ldap\_open — initialize the LDAP library and open a connection to an LDAP server, 347 ldap\_parse\_extended\_result — LDAP message result parser, 349 ldap\_parse\_result — LDAP message result parser, 349 ldap\_parse\_sasl\_bind\_result — LDAP message result parser, 349 ldap\_perror — LDAP protocol error handling functions, 318 ldap\_rename— modify LDAP entry RDN, 345 ldap\_rename\_s — modify LDAP entry RDN, 345 ldap\_result — wait for and return LDAP operation result, 350 ldap\_result2error — LDAP protocol error handling functions, 318

ldap\_sasl\_bind — LDAP bind functions, 299 ldap\_sasl\_bind\_s — LDAP bind functions, 299 ldap\_search — LDAP search operations, 352 ldap\_search\_ext — LDAP search operations, 352 ldap\_search\_ext\_s — LDAP search operations, 352 ldap\_searchprefs — LDAP search preference configuration routeines, 354 ldap\_search\_s — LDAP search operations, 352 ldap\_search\_st — LDAP search operations, 352 ldap\_setfilteraffixes — LDAP filter generating functions, 331 ldap\_set\_rebind\_proc — LDAP bind functions, 299 ldap\_set\_string\_translators — LDAP character set translation functions, 302 ldap\_simple\_bind — LDAP bind functions, 299 ldap\_simple\_bind\_s — LDAP bind functions, 299 ldap\_sort — LDAP entry sorting functions, 356 ldap\_sort\_entries — LDAP entry sorting functions, 356 ldap\_sort\_strcasecmp — LDAP entry sorting functions, 356 ldap\_sort\_values — LDAP entry sorting functions, 356 ldap\_t61\_to\_8859 — LDAP character set translation functions, 302 ldap\_tmplattrs — LDAP display template functions, 309 ldap\_translate\_from\_t61 — LDAP character set translation functions, 302 ldap\_translate\_to\_t61 — LDAP character set translation functions, 302 ldap\_ufn — LDAP user friendly search functions, 358 ldap\_ufn\_search\_c — LDAP user friendly search functions, 358 ldap\_ufn\_search\_ct — LDAP user friendly search functions, 358 ldap\_ufn\_search\_s — LDAP user friendly search functions, 358

ldap\_ufn\_setfilter — LDAP user friendly search functions, 358 ldap\_ufn\_setprefix — LDAP user friendly search functions, 358 ldap\_ufn\_timeout — LDAP user friendly search functions, 358 ldap\_unbind — LDAP bind functions, 299 ldap\_unbind\_ext — LDAP bind functions, 299 ldap\_unbind\_s — LDAP bind functions, 299 ldap\_url — LDAP Uniform Resource Locator functions, 360 ldap\_url\_parse — LDAP Uniform Resource Locator functions, 360 ldap\_url\_search — LDAP Uniform Resource Locator functions, 360 ldap\_url\_search\_s — LDAP Uniform Resource Locator functions, 360 ldap\_url\_search\_st — LDAP Uniform Resource Locator functions, 360 ldap\_vals2text — LDAP entry display functions, 315 library routines for dealing with creation and manipulation of CLIENT handles rpc\_clnt\_create, 448, 499 library routines for client side calls — clnt\_call, 444 - clnt\_freeres, 444 - clnt\_geterr, 444 — clnt\_perrno, 444 - clnt\_perror, 444 - clnt\_sperrno, 444 — clnt\_sperror, 444 - rpc\_broadcast, 444 — rpc\_broadcast\_exp, 444 — rpc\_call, 444 — rpc\_clnt\_calls, 444 library routines for RPC servers — rpc\_svc\_calls, 495 — svc\_dg\_enablecache, 495 — svc\_done, 495 — svc\_exit, 495 - svc\_fdset, 495 — svc\_freeargs, 495 — svc\_getargs, 495 — svc\_getreq\_common, 495 — svc\_getreq\_poll, 495

— svc\_getreqset, 495

library routines for RPC servers (continued) – svc\_getrpccaller, 495 — svc\_max\_pollfd, 495 — svc\_pollfd, 495 — svc\_run, 495 — svc\_sendreply, 495 list mechanisms that support the specified name-type gss\_inquire\_mechs\_for\_name, 250 list the atomic names and references bound in a context — fn\_bindinglist\_destroy, 116 — fn\_bindinglist\_next, 116 — FN\_bindinglist\_t, 116 — fn\_ctx\_list\_bindings, 116 list the atomic names bound in a context — fn\_ctx\_list\_names, 117 — fn\_namelist\_destroy, 117 — fn\_namelist\_next, 117 — FN\_namelist\_t, 117 list the name-types supported by the specified mechanism gss\_inquire\_names\_for\_mech, 252 listen — listen for connections on a socket, 364 look up the link reference bound to a name fn\_ctx\_lookup\_link, 121

#### М

map ASCII mechanism to OID — rpc\_gss\_mech\_to\_oid, 460, 465 map ASCII qop to number — rpc\_gss\_qop\_to\_num, 460, 465 map SLP error codes to messages slp\_strerror, 572 modify LDAP entry RDN — ldap\_modrdn, 345 — ldap\_modrdn2, 345 - ldap\_modrdn2\_s, 345 — ldap\_modrdn\_s, 345 — ldap\_rename, 345 — ldap\_rename\_s, 345 modify multiple attributes associated with named object — fn\_attr\_multi\_modify, 82 modify specified attribute associated with name — fn\_attr\_modify, 73

## Ν

netdir - generic transport name-to-address translation, 367 netdir\_free — generic transport name-to-address translation, 367 netdir\_getbyaddr — generic transport name-to-address translation, 367 netdir\_getbyname — generic transport name-to-address translation, 367 netdir\_mergeaddr — generic transport name-to-address translation, 367 netdir\_options — generic transport name-to-address translation, 367 netdir\_perror — generic transport name-to-address translation, 367 netdir\_sperror — generic transport name-to-address translation, 367 network configuration database entry — endnetconfig, 173 - freenetconfigent, 173 — getnetconfig, 173 - getnetconfigent, 173 -nc\_perror, 173 — nc\_sperror, 173 — setnetconfig, 173 network configuration entry corresponding to NETPATH — endnetpath, 175 — getnetpath, 175 — setnetpath, 175 network entry — endnetent, 170 — getnetbyaddr, 170 — getnetbyaddr\_r, 170 - getnetbyname, 170 getnetbyname\_r, 170 - getnetent, 170 - getnetent\_r, 170 — setnetent, 170 network host entry - endhostent, 157 — gethostbyaddr, 157 gethostbyaddr\_r, 157 — gethostbyname, 157 gethostbyname\_r, 157 - gethostent, 157 — gethostent\_r, 157

network host entry (continued) - sethostent, 157 network listener service format and send listener service request message — nlsrequest, 408 get client's data passed via the listener ---nlsgetcall, 406 nlsprovider, 407 network protocol entry - endprotoent, 180 — getprotobyname, 180 — getprotobyname\_r, 180 — getprotobynumber, 180 — getprotobynumber\_r, 180 - getprotoent, 180 - getprotoent\_r, 180 — setprotoent, 180 NIS+ table functions — nis\_tables nis\_first\_entry, 398 nis modify entry, 398 nis\_next\_entry, 398 nis\_remove\_entry, 398 NIS, change information — yp\_update, 713 NIS client interface — yp\_all, 708 -yp\_bind, 708 — ypclnt, 708 - yperr\_string, 708 — yp\_first, 708 - yp\_get\_default\_domain, 708 — yp\_master, 708 - yp\_match, 708 — yp\_next, 708 — yp\_order, 708 - ypprot\_err, 708 - yp\_unbind, 708 NIS+ error messages nis\_error, 371 nis\_lerror, 371 nis\_perror, 371 nis\_sperrno, 371 nis sperror, 371 nis\_sperror\_r, 371 NIS+ group manipulation functions — nis\_addmember, 372

NIS+ group manipulation functions (continued) — nis\_creategroup, 372 — nis\_destroygroup, 372 -nis\_groups, 372 — nis\_ismember, 372 - nis\_print\_group\_entry, 372 - nis\_removemember, 372 — nis\_verifygroup, 372 NIS+ local names - nis\_freenames, 395 — nis\_getnames, 395 — nis\_local\_directory, 375 -nis\_local\_group, 375 -nis\_local\_host, 375 - nis\_local\_names, 375 — nis\_local\_principal, 375 NIS+ log administration functions — nis\_checkpoint, 392 — nis\_ping, 392 NIS+ miscellaneous functions — nis\_freeservelist, 393 - nis\_freetags, 393 - nis\_getservlist, 393 — nis\_mkdir, 393 — nis\_rmdir, 393 — nis\_server, 393 — nis\_servstate, 393 — nis\_stats, 393 NIS+ namespace functions - nis\_add, 377 -nis\_freeresult, 377 — nis\_lookup, 377 -nis\_modify, 377 — nis\_names, 377 — nis\_remove, 377 NIS+ object formats — nis\_objects, 383 NIS+ subroutines - nis\_clone\_object, 395 - nis\_destroy\_object, 395 — nis\_dir\_cmp, 395 -nis\_domain\_of, 395 — nis\_leaf\_of, 395 - nis\_name\_of, 395 - nis\_print\_object, 395 — nis\_subr, 395

NIS+ table functions

- nis\_add\_entry, 398
- nis\_first\_entry, 398
- nis\_list, 398
- nis\_modify\_entry, 398
- nis\_next\_entry, 398
- nis\_remove\_entry, 398
- nis\_tables, 398
- nis\_tables NIS+ table functions, 398

## 0

obtain information about a credential gss\_inquire\_cred, 246 obtain information about a security context gss\_inquire\_context, 243 obtain per-mechanism information about a credential gss\_inquire\_cred\_by\_mech, 248 open an SLP handle — SLPOpen, 565 overview of the XFN interface — xfn, 694 an overview of XFN attribute operations xfn\_attributes, 695 XFN compound syntax: an overview of XFN model for compound name parsing xfn\_compound\_names, 699

### Ρ

parse service URL — SLPParseSrvURL, 567 pass asynchronous token to security service gss\_process\_context\_token, 255

perform an LDAP add operation

— ldap\_add, 296

- ldap\_add\_ext, 296
- ldap\_add\_ext\_s, 296— ldap\_add\_s, 296

publickey — retrieve public or secret key, 183

#### R

<pre>rac_drop() — remote asynchronous calls,</pre>	474
<pre>rac_poll() — remote asynchronous calls,</pre>	474
<pre>rac_recv() — remote asynchronous calls,</pre>	474

rac\_send() — remote asynchronous calls, 474 rcmd — routines for returning a stream to a remote command, 410 rcmd\_af - routines for returning a stream to a remote command, 410 receive a message from a socket — recv, 412 recvfrom, 412 recvmsg, 412 recv — receive a message from a socket, 412 recvfrom — receive a message from a socket, 412 recvmsg — receive a message from a socket, 412 register an SLP advertisement — SLPReg, 569 release an object identifier gss\_release\_oid, 260 remote procedure calls, library routines for --rpc, 431 remote system return information about users — rusers, rnusers, 513 write to — rstat, 512 write to — rwall, 514 rename the name of a binding fn\_ctx\_rename, 122 res\_hostalias — resolver routines, 423 res\_init - resolver routines, 423 res\_mkquery — resolver routines, 423 res\_nclose — resolver routines, 423 res\_ninit — resolver routines, 423 res\_nmkquery — resolver routines, 423 res\_npquery — resolver routines, 423 res\_nquery — resolver routines, 423 res\_nquerydomain — resolver routines, 423 res\_nsearch — resolver routines, 423 res\_nsend — resolver routines, 423 res\_nsendsigned — resolver routines, 423 resolver — resolver routines, 423 res\_query — resolver routines, 423 res\_querydomain — resolver routines, 423 res\_search — resolver routines, 423 res\_send — resolver routines, 423 res\_update — resolver routines, 423 retrieve public or secret key getpublickey, 183 getsecretkey, 183 publickey, 183

return an attribute's values that matches a specified language subtype ldap\_get\_lang\_values, 333, 429 fn\_ctx\_handle\_from\_initial, 112 return list of configured and discovered scopes — SLPFindScopes, 556 return multiple attributes associated with named object — fn\_attr\_multi\_get, 78 — fn\_multigetlist\_destroy, 78 - fn\_multigetlist\_next, 78 — FN\_multigetlist\_t, 78 return service attributes — SLPFindAttrs, 554 return service URLs - SLPFindSrvs, 558 SLPGetProperty, 563 return specified attribute associated with name — fn\_attr\_get, 66 return the maximum allowed refresh interval -SLPGetRefreshInterval, 564 return values of an attribute - fn\_attr\_get\_values, 68 — fn\_valuelist\_destroy, 68 - fn\_valuelist\_next, 68 — FN\_valuelist\_t, 68 rexec - return stream to a remote command, 429 rexec\_af - return stream to a remote command, 429 rnusers - return information about users on remote machines, 513 routines for returning a stream to a remote command — rcmd, 410 rpc — library routines for remote procedure calls, 431 RPC data transmission using XDR routines xdr, 681 RPC bind service library routines - rpcb\_getaddr, 440 - rpcb\_gettime, 440 — rpcbind, 440 — rpcb\_rmtcall, 440 — rpcb\_set, 440 — rpcb\_unset, 440 - rpc\_getmaps, 440

**RPC** entry — endrpcent, 184 - getrpcbyname, 184 — getrpcbyname\_r, 184 - getrpcbynumber, 184 — getrpcbynumber\_r, 184 - getrpcent, 184 — getrpcent\_r, 184 — setrpcent, 184 RPC library routine for manipulating global RPC attributes for client and server applications — rpc\_control, 455 RPC library routines for registering servers — rpc\_reg, 508 - rpc\_svc\_reg, 508 — svc\_auth\_reg, 508 — svc\_reg, 508 - svc\_unreg, 508 - xprt\_register, 508 — xprt\_unregister, 508 RPC library routines for server side errors - rpc\_svc\_err, 504 — svcerr\_auth, 504 — svcerr\_decode, 504 - svcerr\_noproc, 504 — svcerr\_weakauth, 504 RPC obsolete library routines — authdes\_create, 484 - authunix\_create\_default, 484 — callrpc, 484 — clnt\_broadcast, 484 — clntraw\_create, 484 — clnttcp\_create, 484 - clntudp\_bufcreate, 484 - clntudp\_create, 484 — get\_myaddress, 484 — getrpcport, 484 - pmap\_getmaps, 484 - pmap\_getport, 484 — pmap\_rmtcall, 484 - pmap\_set, 484 - pmap\_unset, 484

— registerrpc, 484

RPC obsolete library routines (continued) – rpc\_soc, 484 — svcfd\_create, 484 — svc\_fds, 484 — svc\_getcaller, 484 — svc\_getreq, 484 - svcraw\_create, 484 — svc\_register, 484 — svctcp\_create, 484 - svcudp\_bufcreate, 484 - svcudp\_create, 484 — svc\_unregister, 484 — xdr\_authunix\_parms, 484 rpc routines rac\_drop() - remote asynchronous calls, 474 474 rac\_poll() — remote asynchronous calls, rac\_recv() — remote asynchronous calls, 474 rac\_send() — remote asynchronous calls, 474 RPC, XDR library routines — rpc\_xdr, 510 — xdr\_accepted\_reply, 510 — xdr\_authsys\_parms, 510 — xdr\_callhdr, 510 — xdr\_callmsg, 510 — xdr\_opaque\_auth, 510 — xdr\_rejected\_reply, 510 — xdr\_replymsg, 510 rpc\_broadcast - library routines for client side calls, 444 rpc\_broadcast\_exp — library routines for client side calls, 444 rpc\_call — library routines for client side calls, 444 rpc\_clnt\_auth — library routines for client side remote procedure call authentication, 442 rpc\_clnt\_calls — library routines for client side calls, 444 Routines, 444, 445 rpc\_clnt\_create — library routines for dealing with creation and manipulation of CLIENT handles, 448

Routines, 449

rpc\_createerr — library routines for dealing with creation and manipulation of CLIENT handles, 448 rpc\_gss\_getcred — get credentials of client, 457 rpc\_gss\_seccreate — create a security context using the RPCSEC\_GSS protocol, 467 rpc — security flavor incorporating GSS-API onto ONC RPC, 478 rpc\_svc\_calls — library routines for RPC servers, 495 Routines, 495 rpc\_svc\_create — library routines for the creation of server handles, 499 rresvport - routines for returning a stream to a remote command, 410 rresvport\_af — routines for returning a stream to a remote command, 410 rstat — get performance data from remote kernel, 512 ruserok — routines for returning a stream to a remote command, 410 rusers - return information about users on remote machines, 513 xdr\_utmpidlearr, 513 rwall — write to specified remote machines, 514

#### S

search for atomic name with specified attributes in single context — fn\_attr\_search, 84 — fn\_searchlist\_destroy, 84 — fn\_searchlist\_next, 84 — FN\_searchlist\_t, 84 search for names whose attributes satisfy filter - fn\_attr\_ext\_search, 59 fn\_ext\_searchlist\_destroy, 59 — fn\_ext\_searchlist\_next, 59 — FN\_ext\_searchlist\_t, 59 send — send message from a socket, 519 sendmsg — send message from a socket, 519 sendto — send message from a socket, 519 Service Access Facility library function

— doconfig, 45

Index 735

Service Location Protocol Application Programming Interface — slp\_api, 537 set an SLP configuration property -SLPSetProperty, 571 set server principal name — rpc\_gss\_set\_svc\_name, 472 setservent — get service entry, 187 shutdown — shut down part of a full-duplex connection, 535 simplified Basic Encoding Rules library encoding functions — ber\_alloc, 27 — ber\_encode, 27 — ber\_printf, 27 — ber\_put\_bitstring, 27 — ber\_put\_boolean, 27 — ber\_put\_int, 27 — ber\_put\_null, 27 - ber\_put\_ostring, 27 - ber\_put\_seq, 27 — ber\_put\_set, 27 - ber\_put\_string, 27 - ber\_start\_seq, 27 — ber\_start\_set, 27 slp\_api — Service Location Protocol Application Programming Interface, 537 SLPClose — close an open SLP handle, 547 SLPDelAttrs — delete attributes, 548 SLPDereg — deregister the SLP advertisement, 550 SLPEscape — escapes SLP reserved characters, 552 SLPFindAttrs — return service attributes, 554 SLPFindScopes - return list of configured and discovered scopes, 556 SLPFindSrvs — return service URLs, 558 SLPFindSrvTypes — find service types, 560 SLPFree — frees memory, 562 SLPGetProperty — return SLP configuration property, 563 SLPGetRefreshInterval — return the maximum allowed refresh interval, 564 SLPOpen — open an SLP handle, 565 SLPParseSrvURL — parse service URL, 567 SLPReg — register an SLP advertisement, 569 SLPSetProperty — set an SLP configuration property, 571

slp\_strerror — map SLP error codes to messages, 572 SLPUnescape — translate escaped characters into UTF-8, 573 socket - create an endpoint for communication, 575 socket accept a connection — accept, 18 bind a name — bind, 31 get options — getsocketopt, 193 get name — getsockname, 191 get name of connected peer getpeername, 177 initiate a connection — connect, 36 listen for connections — listen, 364 send message from - send, sendto, sendmsg, 519 set options — setsocketopt, 193 shut down part of a full-duplex connection — shutdown, 535 socketpair — create a pair of connected sockets, 580 spray — scatter data in order to test the network, 583 step through LDAP entry attributes – ldap\_first\_attribute, 322 — ldap\_next\_attribute, 322 STREAMS accept a connection on a socket accept, 18 bind a name to a socket — bind, 31 create a pair of connected sockets socketpair, 580 socket, 575 get and set socket options - getsockopt, setsockopt, 193 get name of peer connected to socket ---getpeername, 177 get socket name — getsockname, 191 initiate a connection on a socket connect, 36 listen for connections on a socket ---listen, 364 send a message from a socket - send, sendto, sendmsg, 519

STREAMS (continued) shut down part of a full-duplex connection – shutdown, 535 svc\_control — library routines for the creation of server handles, 499 svc\_create — library routines for the creation of server handles, 499 svc\_destroy — library routines for the creation of server handles, 499 svc\_dg\_create — library routines for the creation of server handles, 499 svc\_dg\_enablecache — library routines for RPC servers, 495 svc\_done — library routines for RPC servers, 495 svc\_door\_create — library routines for the creation of server handles, 499 svc\_exit — library routines for RPC servers, 495 svc\_fd\_create — library routines for the creation of server handles, 499 svc\_fdset — library routines for RPC servers, 495 svc\_freeargs — library routines for RPC servers, 495 svc\_getargs — library routines for RPC servers, 495 svc\_getreq\_common — library routines for RPC servers, 495 svc\_getreq\_poll — library routines for RPC servers, 495 svc\_getreqset — library routines for RPC servers, 495 svc\_getrpccaller — library routines for RPC servers, 495 svc\_max\_pollfd — library routines for RPC servers, 495 svc\_pollfd — library routines for RPC servers, 495 svc\_raw\_create - library routines for the creation of server handles, 499 svc\_run — library routines for RPC servers, 495 svc\_sendreply — library routines for RPC servers, 495

svc\_tli\_create — library routines for the creation
 of server handles, 499

svc\_tp\_create — library routines for the creation
 of server handles, 499

svc\_vc\_create — library routines for the creation
 of server handles, 499

#### Т

taddr2uaddr — generic transport name-to-address translation, 367 t\_alloc — allocate memory for argument structures, 589 t\_bind — bind an address to a transport endpoint, 592 terminal line establish an outgoing connection — dial, 43 transfer a security context to another process gss\_export\_sec\_context, 227 translate between node name and address getaddrinfo, 153 translate escaped characters into UTF-8 — SLPUnescape, 573 transport functions allocate memory, 589

#### U

uaddr2taddr — generic transport name-to-address translation, 367 users return information from remote machines rusers, rnusers, 513

### V

verify a message with attached cryptographic message — gss\_wrap, 265 verify integrity of a received message gss\_verify\_mic, 267

### W

wait for and return LDAP operation result — ldap\_result, 350

## Х

XDR library routines — xdr, 681 — xdr\_admin, 683 — xdr\_control, 683 - xdr\_getpos, 683  $- xdr_{inline}$ , 683 - xdrrec\_endofrecord, 683 — xdrrec\_eof, 683 — xdrrec\_readbytes, 683 — xdrrec\_skiprecord, 683 — xdr\_setpos, 683 — xdr\_sizeof, 683 XDR library routines for complex data structures  $-xdr_array$ , 685 — xdr\_bytes, 685 — xdr\_complex, 685 — xdr\_opaque, 685 — xdr\_pointer, 685 — xdr\_reference, 685 — xdr\_string, 685 — xdr\_union, 685 — xdr\_vector, 685 - xdr\_wrapstring, 685 XDR library routines for RPC — rpc\_xdr, 510 — xdr\_accepted\_reply, 510 - xdr\_authsys\_parms, 510 — xdr\_callhdr, 510 — xdr\_callmsg, 510 — xdr\_opaque\_auth, 510 - xdr\_rejected\_reply, 510 — xdr\_replymsg, 510 XDR library routines for simple data structures — xdr\_bool, 690 — xdr\_char, 690 — xdr\_double, 690 — xdr\_enum, 690 - xdr\_float, 690 — xdr\_free, 690 — xdr\_hyper, 690 — xdr\_int, 690 — xdr\_long, 690 — xdr\_longlong\_t, 690 — xdr\_quadruple, 690 — xdr\_short, 690

XDR library routines for simple data structures (continued) — xdr\_simple, 690 — xdr\_u\_char, 690 — xdr\_u\_hyper, 690 — xdr\_u\_int, 690 — xdr\_u\_long, 690 — xdr\_u\_longlong\_t, 690  $- xdr_u_short, 690$ — xdr\_void, 690 XDR stream creation library routines — xdr\_create, 688 — xdr\_destroy, 688 - xdrmem\_create, 688 - xdrrec\_create, 688 — xdrstdio\_create, 688 xdr\_statstime - get performance data from remote kernel, 512 xdr\_statsvar — get performance data from remote kernel, 512 xfn — overview of the XFN interface, 694 XFN attribute - fn\_attribute\_add, 71 — fn\_attribute\_assign, 71 — fn\_attribute\_copy, 71 - fn\_attribute\_create, 71 - fn\_attribute\_destroy, 71 — fn\_attribute\_first, 71 - fn\_attribute\_identifier, 71 - fn\_attribute\_next, 71 - fn\_attribute\_remove, 71 - fn\_attribute\_syntax, 71 - FN\_attribute\_t, 71 - fn\_attribute\_valuecount, 71 XFN attributes, a set of — fn\_attrset\_add, 89 — fn\_attrset\_assign, 89 — fn\_attrset\_copy, 89 - fn\_attrset\_count, 89 - fn\_attrset\_create, 89 fn\_attrset\_destroy, 89 - fn\_attrset\_first, 89 - fn\_attrset\_get, 89 - fn\_attrset\_next, 89 - fn\_attrset\_remove, 89

— FN\_attrset\_t, 89

XFN compound name

- fn\_compound\_name\_append\_comp, 98
- fn\_compound\_name\_assign, 98
- fn\_compound\_name\_copy, 98
- fn\_compound\_name\_count, 98
- fn\_compound\_name\_delete\_all, 98
- fn\_compound\_name\_delete\_comp, 98
- fn\_compound\_name\_destroy, 98
- fn\_compound\_name\_first, 98

fn\_compound\_name\_from\_syntax\_attrs, 98 \_\_\_

- fn\_compound\_name\_get\_syntax\_attrs, 98
- fn\_compound\_name\_insert\_comp, 98
- fn\_compound\_name\_is\_empty, 98
- fn\_compound\_name\_is\_equal, 98
- fn\_compound\_name\_is\_prefix, 98
- fn\_compound\_name\_is\_suffix, 98
- fn\_compound\_name\_last, 98
- fn\_compound\_name\_next, 98
- fn\_compound\_name\_prefix, 98
- fn\_compound\_name\_prepend\_comp, 98
- fn\_compound\_name\_prev, 98
- fn\_compound\_name\_suffix, 98
- FN\_compound\_name\_t, 98
- fn\_string\_from\_compound\_name, 98
- an XFN context FN\_ctx\_t, 125
- an XFN identifier FN\_identifier\_t, 128 XFN reference
  - fn\_ref\_addrcount, 131
  - fn\_ref\_append\_addr, 131
  - fn\_ref\_assign, 131
  - fn\_ref\_copy, 131
  - fn\_ref\_create, 131
  - fn\_ref\_create\_link, 131
  - fn\_ref\_delete\_addr, 131
  - fn\_ref\_delete\_all, 131
  - fn\_ref\_description, 131
  - fn\_ref\_destroy, 131
  - fn\_ref\_first, 131
  - fn\_ref\_insert\_addr, 131
  - fn\_ref\_is\_link, 131
  - fn\_ref\_link\_name, 131
  - fn\_ref\_next, 131
  - fn\_ref\_prepend\_addr, 131
  - FN\_ref\_t, 131
  - fn\_ref\_type, 131

XFN status object — fn\_status\_advance\_by\_name, 145 — fn\_status\_append\_remaining\_name, 145 fn\_status\_append\_resolved\_name, 145 - fn\_status\_assign, 145 — fn\_status\_code, 145 - fn\_status\_copy, 145 - fn\_status\_create, 145 — fn\_status\_description, 145 — fn\_status\_destroy, 145 - fn\_status\_diagnostic\_message, 145 fn\_status\_is\_success, 145 — fn\_status\_link\_code, 145 fn\_status\_link\_diagnostic\_message, 145 - fn\_status\_link\_remaining\_name, 145 - fn\_status\_link\_resolved\_name, 145 fn\_status\_link\_resolved\_ref, 145 - fn\_status\_remaining\_name, 145 - fn\_status\_resolved\_name, 145 — fn\_status\_resolved\_ref, 145 - fn\_status\_set, 145 - fn\_status\_set\_code, 145 fn\_status\_set\_diagnostic\_message, 145 — fn\_status\_set\_link\_code, 145 fn\_status\_set\_link\_diagnostic\_message, 145 - fn\_status\_set\_link\_remaining\_name, 145 - fn\_status\_set\_link\_resolved\_name, 145 fn\_status\_set\_link\_resolved\_ref, 145 - fn\_status\_set\_remaining\_name, 145 fn\_status\_set\_resolved\_name, 145 fn\_status\_set\_resolved\_ref, 145 fn\_status\_set\_success, 145 — FN\_status\_t, 145 xfn\_attributes — an overview of XFN attribute operations, 695 xfn\_compound\_names — XFN compound syntax: an overview of XFN model for compound name parsing, 699 xfn\_status\_codes — descriptions of XFN status codes, 704

XFN Status Codes, 704