



man pages section 3: Extended Library Functions

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

Part No: 817-3941-10
September 2004

Copyright 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

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

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

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

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

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

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 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

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

Certaines 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 de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

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

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



040714 @ 9061



Contents

Preface 15

Extended Library Functions 21

aclcheck(3SEC) 22

aclsort(3SEC) 24

acltomode(3SEC) 25

acltotext(3SEC) 26

acos(3M) 28

acosh(3M) 29

asin(3M) 30

atan2(3M) 31

atan(3M) 32

au_open(3BSM) 33

au_preselect(3BSM) 35

au_to(3BSM) 37

au_user_mask(3BSM) 41

bgets(3GEN) 43

bufsplit(3GEN) 45

cbrt(3M) 46

ceil(3M) 47

config_admin(3CFGADM) 48

ConnectToServer(3DMI) 56

copylist(3GEN) 57

copysign(3M) 58

cos(3M) 59

cosh(3M) 60

cpc(3CPC) 61
 cpc_access(3CPC) 64
 cpc_bind_event(3CPC) 65
 cpc_count_usr_events(3CPC) 71
 cpc_event(3CPC) 73
 cpc_event_diff(3CPC) 75
 cpc_getcpuver(3CPC) 77
 cpc_pctx_bind_event(3CPC) 79
 cpc_seterrfn(3CPC) 81
 cpc_shared_open(3CPC) 82
 cpc_strtoevent(3CPC) 84
 cpc_version(3CPC) 87
 demangle(3EXT) 88
 devid_get(3DEVID) 89
 di_binding_name(3DEVINFO) 93
 di_child_node(3DEVINFO) 95
 di_devfs_path(3DEVINFO) 97
 di_init(3DEVINFO) 98
 di_minor_devt(3DEVINFO) 101
 di_minor_next(3DEVINFO) 102
 di_prom_init(3DEVINFO) 103
 di_prom_prop_data(3DEVINFO) 104
 di_prom_prop_lookup_bytes(3DEVINFO) 106
 di_prop_bytes(3DEVINFO) 108
 di_prop_lookup_bytes(3DEVINFO) 111
 di_prop_next(3DEVINFO) 113
 DisconnectToServer(3DMI) 114
 di_walk_minor(3DEVINFO) 115
 di_walk_node(3DEVINFO) 117
 DmiAddComponent(3DMI) 118
 DmiAddRow(3DMI) 122
 dmi_error(3DMI) 127
 DmiGetConfig(3DMI) 128
 DmiListAttributes(3DMI) 131
 DmiRegisterCi(3DMI) 136
 ea_error(3EXACCT) 138
 ea_open(3EXACCT) 139
 ea_pack_object(3EXACCT) 141

ea_set_item(3EXACCT)	146
efi_alloc_and_init(3EXT)	149
elf32_checksum(3ELF)	151
elf32_fsize(3ELF)	152
elf32_getehdr(3ELF)	153
elf32_getphdr(3ELF)	155
elf32_getshdr(3ELF)	157
elf32_xlatetof(3ELF)	159
elf(3ELF)	161
elf_begin(3ELF)	167
elf_cntl(3ELF)	172
elf_errmsg(3ELF)	174
elf_fill(3ELF)	175
elf_flagdata(3ELF)	176
elf_getarhdr(3ELF)	178
elf_getarsym(3ELF)	180
elf_getbase(3ELF)	181
elf_getdata(3ELF)	182
elf_getident(3ELF)	187
elf_getscn(3ELF)	189
elf_hash(3ELF)	191
elf_kind(3ELF)	192
elf_rawfile(3ELF)	193
elf_strptr(3ELF)	195
elf_update(3ELF)	196
elf_version(3ELF)	200
erf(3M)	201
exp(3M)	202
expm1(3M)	203
fabs(3M)	204
floor(3M)	205
fmod(3M)	206
freeDmiString(3DMI)	207
gelf(3ELF)	208
getacinfo(3BSM)	213
getauclassent(3BSM)	215
getauditflags(3BSM)	217
getauevent(3BSM)	218

getauthattr(3SECDB) 220
 getauusernam(3BSM) 223
 getddent(3BSM) 225
 getdmapent(3BSM) 227
 getexecattr(3SECDB) 229
 getfauditflags(3BSM) 232
 getprofattr(3SECDB) 233
 getprojent(3PROJECT) 235
 getuserattr(3SECDB) 239
 gmatch(3GEN) 241
 hypot(3M) 242
 idn_decodename(3EXT) 243
 IFDHCcloseChannel(3SMARTCARD) 251
 IFDHControl(3SMARTCARD) 252
 IFDHCreateChannel(3SMARTCARD) 253
 IFDHCreateChannelByName(3SMARTCARD) 254
 IFDHGetCapabilities(3SMARTCARD) 256
 IFDHICCPresence(3SMARTCARD) 258
 IFDHPowerICC(3SMARTCARD) 259
 IFDHSetCapabilities(3SMARTCARD) 261
 IFDHSetProtocolParameters(3SMARTCARD) 262
 IFDHTransmitToICC(3SMARTCARD) 264
 ilogb(3M) 266
 isencrypt(3GEN) 267
 isnan(3M) 268
 j0(3M) 269
 kstat(3KSTAT) 270
 kstat_chain_update(3KSTAT) 276
 kstat_lookup(3KSTAT) 277
 kstat_open(3KSTAT) 278
 kstat_read(3KSTAT) 279
 kva_match(3SECDB) 280
 kvm_getu(3KVM) 281
 kvm_nextproc(3KVM) 283
 kvm_nlist(3KVM) 285
 kvm_open(3KVM) 286
 kvm_read(3KVM) 288
 ld_support(3EXT) 290

lgamma(3M) 291
 lgrp_affinity_get(3LGRP) 293
 lgrp_children(3LGRP) 295
 lgrp_cookie_stale(3LGRP) 296
 lgrp_cpus(3LGRP) 297
 lgrp_fini(3LGRP) 298
 lgrp_home(3LGRP) 299
 lgrp_init(3LGRP) 300
 lgrp_latency(3LGRP) 301
 lgrp_mem_size(3LGRP) 302
 lgrp_nlgrps(3LGRP) 303
 lgrp_parents(3LGRP) 304
 lgrp_root(3LGRP) 305
 lgrp_version(3LGRP) 306
 lgrp_view(3LGRP) 307
 libdevinfo(3DEVINFO) 308
 libnvpair(3NVPAIR) 311
 libpicl(3PICL) 312
 libpicltree(3PICLTREE) 315
 libtnfctl(3TNF) 318
 log10(3M) 323
 log1p(3M) 324
 log(3M) 325
 logb(3M) 326
 maillock(3MAIL) 327
 matherr(3M) 329
 m_create_layout(3LAYOUT) 335
 md5(3EXT) 337
 m_destroy_layout(3LAYOUT) 339
 media_findname(3VOLMGT) 340
 media_getattr(3VOLMGT) 343
 media_getid(3VOLMGT) 345
 m_getvalues_layout(3LAYOUT) 346
 mkdirp(3GEN) 347
 mp(3MP) 349
 m_setvalues_layout(3LAYOUT) 351
 m_transform_layout(3LAYOUT) 352
 m_wtransform_layout(3LAYOUT) 357

newDmiOctetString(3DMI) 363
 newDmiString(3DMI) 364
 nextafter(3M) 365
 nlist(3ELF) 366
 NOTE(3EXT) 367
 nvlist_add_boolean(3NVPAIR) 369
 nvlist_alloc(3NVPAIR) 371
 nvlist_lookup_boolean(3NVPAIR) 376
 nvlist_next_nvpair(3NVPAIR) 378
 nvlist_remove(3NVPAIR) 380
 nvpair_value_byte(3NVPAIR) 381
 p2open(3GEN) 383
 pam(3PAM) 385
 pam_acct_mgmt(3PAM) 388
 pam_authenticate(3PAM) 389
 pam_chauthtok(3PAM) 391
 pam_getenv(3PAM) 393
 pam_getenvlist(3PAM) 394
 pam_get_user(3PAM) 395
 pam_open_session(3PAM) 397
 pam_putenv(3PAM) 399
 pam_setcred(3PAM) 401
 pam_set_data(3PAM) 403
 pam_set_item(3PAM) 405
 pam_sm(3PAM) 407
 pam_sm_acct_mgmt(3PAM) 411
 pam_sm_authenticate(3PAM) 413
 pam_sm_chauthtok(3PAM) 415
 pam_sm_open_session(3PAM) 418
 pam_sm_setcred(3PAM) 420
 pam_start(3PAM) 422
 pam_strerror(3PAM) 425
 pathfind(3GEN) 426
 pctx_capture(3CPC) 428
 pctx_set_events(3CPC) 430
 picld_log(3PICLTREE) 433
 picld_plugin_register(3PICLTREE) 434
 picl_get_first_prop(3PICL) 436

picl_get_next_by_row(3PICL) 438
 picl_get_prop_by_name(3PICL) 439
 picl_get_propinfo(3PICL) 440
 picl_get_propinfo_by_name(3PICL) 441
 picl_get_propval(3PICL) 442
 picl_get_root(3PICL) 444
 picl_initialize(3PICL) 445
 picl_set_propval(3PICL) 446
 picl_shutdown(3PICL) 448
 picl_strerror(3PICL) 449
 picl_wait(3PICL) 450
 picl_walk_tree_by_class(3PICL) 451
 pool_associate(3POOL) 452
 pool_component_info(3POOL) 455
 pool_component_to_elem(3POOL) 457
 pool_conf_alloc(3POOL) 458
 pool_dynamic_location(3POOL) 463
 pool_error(3POOL) 465
 pool_get_binding(3POOL) 467
 pool_get_pool(3POOL) 470
 pool_get_property(3POOL) 472
 pool_resource_create(3POOL) 475
 pool_value_alloc(3POOL) 479
 pool_walk_components(3POOL) 482
 pow(3M) 484
 printDmiAttributeValues(3DMI) 485
 printDmiDataUnion(3DMI) 486
 printDmiString(3DMI) 487
 project_walk(3PROJECT) 488
 ptree_add_node(3PICLTREE) 490
 ptree_add_prop(3PICLTREE) 491
 ptree_create_and_add_node(3PICLTREE) 492
 ptree_create_and_add_prop(3PICLTREE) 493
 ptree_create_node(3PICLTREE) 494
 ptree_create_prop(3PICLTREE) 496
 ptree_create_table(3PICLTREE) 498
 ptree_find_node(3PICLTREE) 499
 ptree_get_first_prop(3PICLTREE) 500

ptree_get_next_by_row(3PICLTREE) 501
 ptree_get_node_by_path(3PICLTREE) 502
 ptree_get_prop_by_name(3PICLTREE) 504
 ptree_get_propinfo(3PICLTREE) 505
 ptree_get_propinfo_by_name(3PICLTREE) 506
 ptree_get_propval(3PICLTREE) 507
 ptree_get_root(3PICLTREE) 508
 ptree_init_propinfo(3PICLTREE) 509
 ptree_post_event(3PICLTREE) 510
 ptree_register_handler(3PICLTREE) 511
 ptree_unregister_handler(3PICLTREE) 512
 ptree_update_propval(3PICLTREE) 513
 ptree_walk_tree_by_class(3PICLTREE) 514
 read_vtoc(3EXT) 515
 reg_ci_callback(3DMI) 516
 regexpr(3GEN) 517
 remainder(3M) 520
 rint(3M) 521
 rsm_create_localmemory_handle(3RSM) 522
 rsm_get_controller(3RSM) 524
 rsm_get_interconnect_topology(3RSM) 526
 rsm_get_segmentid_range(3RSM) 528
 rsm_intr_signal_post(3RSM) 530
 rsm_memseg_export_create(3RSM) 532
 rsm_memseg_export_publish(3RSM) 535
 rsm_memseg_get_pollfd(3RSM) 538
 rsm_memseg_import_connect(3RSM) 539
 rsm_memseg_import_get(3RSM) 541
 rsm_memseg_import_init_barrier(3RSM) 543
 rsm_memseg_import_map(3RSM) 544
 rsm_memseg_import_open_barrier(3RSM) 546
 rsm_memseg_import_put(3RSM) 548
 rsm_memseg_import_putv(3RSM) 550
 rsm_memseg_import_set_mode(3RSM) 552
 rtld_audit(3EXT) 553
 rtld_db(3EXT) 554
 scalb(3M) 555
 scalbn(3M) 556

SCF_Card_exchangeAPDU(3SMARTCARD) 557
 SCF_Card_lock(3SMARTCARD) 559
 SCF_Card_reset(3SMARTCARD) 562
 SCF_Session_close(3SMARTCARD) 564
 SCF_Session_freeInfo(3SMARTCARD) 566
 SCF_Session_getInfo(3SMARTCARD) 568
 SCF_Session_getSession(3SMARTCARD) 572
 SCF_Session_getTerminal(3SMARTCARD) 574
 SCF_strerror(3SMARTCARD) 577
 SCF_Terminal_addEventListener(3SMARTCARD) 578
 SCF_Terminal_getCard(3SMARTCARD) 584
 SCF_Terminal_waitForCardPresent(3SMARTCARD) 586
 sendfile(3EXT) 589
 sendfilev(3EXT) 592
 setproject(3PROJECT) 595
 significand(3M) 597
 sin(3M) 598
 sinh(3M) 599
 sqrt(3M) 600
 SSAgentIsAlive(3SNMP) 601
 SSAOidCmp(3SNMP) 604
 SSAStrCpy(3SNMP) 606
 strccpy(3GEN) 607
 strfind(3GEN) 609
 sysevent_bind_handle(3SYSEVENT) 610
 sysevent_free(3SYSEVENT) 612
 sysevent_get_attr_list(3SYSEVENT) 613
 sysevent_get_class_name(3SYSEVENT) 614
 sysevent_get_vendor_name(3SYSEVENT) 616
 sysevent_post_event(3SYSEVENT) 618
 sysevent_subscribe_event(3SYSEVENT) 620
 tan(3M) 623
 tanh(3M) 624
 tnfctl_buffer_alloc(3TNF) 625
 tnfctl_close(3TNF) 627
 tnfctl_indirect_open(3TNF) 629
 tnfctl_internal_open(3TNF) 632
 tnfctl_kernel_open(3TNF) 634

tnftcl_pid_open(3TNF) 635
 tnftcl_probe_apply(3TNF) 640
 tnftcl_probe_state_get(3TNF) 643
 tnftcl_register_funcs(3TNF) 647
 tnftcl_strerror(3TNF) 648
 tnftcl_trace_attrs_get(3TNF) 649
 tnftcl_trace_state_set(3TNF) 651
 TNF_DECLARE_RECORD(3TNF) 653
 TNF_PROBE(3TNF) 656
 tnft_process_disable(3TNF) 661
 tracing(3TNF) 663
 volmgt_acquire(3VOLMGT) 667
 volmgt_check(3VOLMGT) 670
 volmgt_feature_enabled(3VOLMGT) 672
 volmgt_inuse(3VOLMGT) 673
 volmgt_ownspath(3VOLMGT) 674
 volmgt_release(3VOLMGT) 675
 volmgt_root(3VOLMGT) 676
 volmgt_running(3VOLMGT) 677
 volmgt_symname(3VOLMGT) 678
 wsreg_add_child_component(3WSREG) 680
 wsreg_add_compatible_version(3WSREG) 682
 wsreg_add_dependent_component(3WSREG) 684
 wsreg_add_display_name(3WSREG) 686
 wsreg_add_required_component(3WSREG) 688
 wsreg_can_access_registry(3WSREG) 690
 wsreg_clone_component(3WSREG) 692
 wsreg_components_equal(3WSREG) 693
 wsreg_create_component(3WSREG) 694
 wsreg_get(3WSREG) 695
 wsreg_initialize(3WSREG) 696
 wsreg_query_create(3WSREG) 697
 wsreg_query_set_id(3WSREG) 698
 wsreg_query_set_instance(3WSREG) 699
 wsreg_query_set_location(3WSREG) 700
 wsreg_query_set_unique_name(3WSREG) 701
 wsreg_query_set_version(3WSREG) 702
 wsreg_register(3WSREG) 703

wsreg_set_data(3WSREG) 705
wsreg_set_id(3WSREG) 707
wsreg_set_instance(3WSREG) 708
wsreg_set_location(3WSREG) 710
wsreg_set_parent(3WSREG) 711
wsreg_set_type(3WSREG) 712
wsreg_set_uninstaller(3WSREG) 713
wsreg_set_unique_name(3WSREG) 714
wsreg_set_vendor(3WSREG) 715
wsreg_set_version(3WSREG) 716
wsreg_unregister(3WSREG) 717
y0(3M) 719

Index 721

Preface

Both novice users and those familiar 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 commands or functions documented, followed by a brief description of what they do.								
SYNOPSIS	This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required. The following special characters are used in this section: <table> <tr> <td>[]</td><td>Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.</td></tr> <tr> <td>. . .</td><td>Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename . . .".</td></tr> <tr> <td> </td><td>Separator. Only one of the arguments separated by this character can be specified at a time.</td></tr> <tr> <td>{ }</td><td>Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.</td></tr> </table>	[]	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 . . .".		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.
[]	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 . . .".								
	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 <code>ioctl(2)</code> system call is called <code>ioctl</code> and generates its own heading. <code>ioctl</code> calls for a specific device are listed alphabetically (on the man page for that specific device). <code>ioctl</code> calls are used for a particular class of devices all of which have an <code>io</code> ending, such as <code>mtio(7I)</code> .
OPTIONS	This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable <code>errno</code> 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 <code>example%</code>, or if the user must be superuser, <code>example#</code>. 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 <code>attributes(5)</code> for more information.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.

DIAGNOSTICS

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

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.

Extended Library Functions

aclcheck(3SEC)

NAME	aclcheck – check the validity of an ACL						
SYNOPSIS	<pre>cc [flag...] file... -lsec [library...] #include <sys/acl.h> int aclcheck(aclent_t *aclbufp, int nentries, int *which);</pre>						
DESCRIPTION	The <code>aclcheck()</code> function checks the validity of an ACL pointed to by <code>aclbufp</code>. The <code>nentries</code> argument is the number of entries contained in the buffer. The <code>which</code> parameter returns the index of the first entry that is invalid. The function verifies that an ACL pointed to by <code>aclbufp</code> is valid according to the following rules: ■ There must be exactly one <code>GROUP_OBJ</code> ACL entry. ■ There must be exactly one <code>USER_OBJ</code> ACL entry. ■ There must be exactly one <code>OTHER_OBJ</code> ACL entry. ■ If there are any <code>GROUP</code> ACL entries, then the group ID in each group ACL entry must be unique. ■ If there are any <code>USER</code> ACL entries, then the user ID in each user ACL entry must be unique. ■ If there are any <code>GROUP</code> or <code>USER</code> ACL entries, then there must be exactly one <code>CLASS_OBJ</code> (ACL mask) entry. ■ If there are any default ACL entries, then the following apply: ■ There must be exactly one default <code>GROUP_OBJ</code> ACL entry. ■ There must be exactly one default <code>OTHER_OBJ</code> ACL entry. ■ There must be exactly one default <code>USER_OBJ</code> ACL entry. ■ If there are any <code>DEF_GROUP</code> entries, then the group ID in each <code>DEF_GROUP</code> ACL entry must be unique. ■ If there are any <code>DEF_USER</code> entries, then the user ID in each <code>DEF_USER</code> ACL entry must be unique. ■ If there are any <code>DEF_GROUP</code> or <code>DEF_USER</code> entries, then there must be exactly one <code>DEF_CLASS_OBJ</code> (default ACL mask) entry. ■ If any of the above rules are violated, then the function fails with <code>errno</code> set to <code>EINVAL</code>.						
RETURN VALUES	If the ACL is valid, <code>aclcheck()</code> will return 0. Otherwise <code>errno</code> is set to <code>EINVAL</code> and return code is set to one of the following: <table><tr><td><code>GRP_ERROR</code></td><td>There is more than one <code>GROUP_OBJ</code> or <code>DEF_GROUP_OBJ</code> ACL entry.</td></tr><tr><td><code>USER_ERROR</code></td><td>There is more than one <code>USER_OBJ</code> or <code>DEF_USER_OBJ</code> ACL entry.</td></tr><tr><td><code>CLASS_ERROR</code></td><td>There is more than one <code>CLASS_OBJ</code> (ACL mask) or <code>DEF_CLASS_OBJ</code> (default ACL mask) entry.</td></tr></table>	<code>GRP_ERROR</code>	There is more than one <code>GROUP_OBJ</code> or <code>DEF_GROUP_OBJ</code> ACL entry.	<code>USER_ERROR</code>	There is more than one <code>USER_OBJ</code> or <code>DEF_USER_OBJ</code> ACL entry.	<code>CLASS_ERROR</code>	There is more than one <code>CLASS_OBJ</code> (ACL mask) or <code>DEF_CLASS_OBJ</code> (default ACL mask) entry.
<code>GRP_ERROR</code>	There is more than one <code>GROUP_OBJ</code> or <code>DEF_GROUP_OBJ</code> ACL entry.						
<code>USER_ERROR</code>	There is more than one <code>USER_OBJ</code> or <code>DEF_USER_OBJ</code> ACL entry.						
<code>CLASS_ERROR</code>	There is more than one <code>CLASS_OBJ</code> (ACL mask) or <code>DEF_CLASS_OBJ</code> (default ACL mask) entry.						

aclcheck(3SEC)

OTHER_ERROR	There is more than one OTHER_OBJ or DEF_OTHER_OBJ ACL entry.
DUPLICATE_ERROR	Duplicate entries of USER, GROUP, DEF_USER, or DEF_GROUP.
ENTRY_ERROR	The entry type is invalid.
MISS_ERROR	Missing an entry. The <i>which</i> parameter returns -1 in this case.
MEM_ERROR	The system cannot allocate any memory. The <i>which</i> parameter returns -1 in this case.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Unsafe

SEE ALSO acl(2), aclsort(3SEC), attributes(5)

aclsort(3SEC)

NAME	aclsort – sort an ACL						
SYNOPSIS	<pre>cc [flag ...] file ... -lsec [library ...] #include <sys/acl.h> int aclsort(int <i>nentries</i>, int <i>calclass</i>, aclent_t *<i>aclbufp</i>);</pre>						
DESCRIPTION	The <i>aclbufp</i> argument points to a buffer containing ACL entries. The <i>nentries</i> argument specifies the number of ACL entries in the buffer. The <i>calclass</i> argument, if non-zero, indicates that the CLASS_OBJ (ACL mask) permissions should be recalculated. The union of the permission bits associated with all ACL entries in the buffer other than CLASS_OBJ, OTHER_OBJ, and USER_OBJ is calculated. The result is copied to the permission bits associated with the CLASS_OBJ entry. The <code>aclsort()</code> function sorts the contents of the ACL buffer as follows: ■ Entries will be in the order USER_OBJ, USER, GROUP_OBJ, GROUP, CLASS_OBJ (ACL mask), OTHER_OBJ, DEF_USER_OBJ, DEF_USER, DEF_GROUP_OBJ, DEF_GROUP, DEF_CLASS_OBJ (default ACL mask), and DEF_OTHER_OBJ. ■ Entries of type USER, GROUP, DEF_USER, and DEF_GROUP will be sorted in increasing order by ID. The <code>aclsort()</code> function will succeed if all of the following are true: ■ There is exactly one entry each of type USER_OBJ, GROUP_OBJ, CLASS_OBJ (ACL mask), and OTHER_OBJ. ■ There is exactly one entry each of type DEF_USER_OBJ, DEF_GROUP_OBJ, DEF_CLASS_OBJ (default ACL mask), and DEF_OTHER_OBJ if there are any default entries. ■ Entries of type USER, GROUP, DEF_USER, or DEF_GROUP may not contain duplicate entries. A duplicate entry is one of the same type containing the same numeric ID.						
RETURN VALUES	Upon successful completion, the the function returns 0. Otherwise, it returns -1.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Unsafe						
SEE ALSO	<code>acl(2)</code> , <code>aclcheck(3SEC)</code> , <code>attributes(5)</code>						

NAME	acltomode, aclfrommode – convert an ACL to or from permission bits						
SYNOPSIS	<pre>cc [flag...] file... -lsec [library...] #include <sys/types.h> #include <sys/acl.h> int acltomode(aclent_t *aclbufp, int nentries, mode_t *modep); int aclfrommode(aclent_t *aclbufp, int nentries, mode_t *modep);</pre>						
DESCRIPTION	The <code>acltomode()</code> function converts an ACL pointed to by <code>aclbufp</code> into the permission bits buffer pointed to by <code>modep</code>. If the <code>USER_OBJ</code> ACL entry, <code>GROUP_OBJ</code> ACL entry, or the <code>OTHER_OBJ</code> ACL entry cannot be found in the ACL buffer, then the function fails with <code>errno</code> set to <code>EINVAL</code>. The <code>USER_OBJ</code> ACL entry permission bits are copied to the file owner class bits in the permission bits buffer. The <code>OTHER_OBJ</code> ACL entry permission bits are copied to the file other class bits in the permission bits buffer. If there is a <code>CLASS_OBJ</code> (ACL mask) entry, the <code>CLASS_OBJ</code> ACL entry permission bits are copied to the file group class bits in the permission bits buffer. Otherwise, the <code>GROUP_OBJ</code> ACL entry permission bits are copied to the file group class bits in the permission bits buffer. The <code>aclfrommode()</code> function converts the permission bits pointed to by <code>modep</code> into an ACL pointed to by <code>aclbufp</code>. If the <code>USER_OBJ</code> ACL entry, <code>GROUP_OBJ</code> ACL entry, or the <code>OTHER_OBJ</code> ACL entry cannot be found in the ACL buffer, the function fails with <code>errno</code> set to <code>EINVAL</code>. The file owner class bits from the permission bits buffer are copied to the <code>USER_OBJ</code> ACL entry. The file other class bits from the permission bits buffer are copied to the <code>OTHER_OBJ</code> ACL entry. If there is a <code>CLASS_OBJ</code> (ACL mask) entry, the file group class bits from the permission bits buffer are copied to the <code>CLASS_OBJ</code> ACL entry, and the <code>GROUP_OBJ</code> ACL entry is not modified. Otherwise, the file group class bits from the permission bits buffer are copied to the <code>GROUP_OBJ</code> ACL entry. The <code>nentries</code> argument represents the number of ACL entries in the buffer pointed to by <code>aclbufp</code>.						
RETURN VALUES	Upon successful completion, the function returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate the error.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Unsafe						
SEE ALSO	<code>acl(2)</code> , <code>attributes(5)</code>						

acltotext(3SEC)

NAME	acltotext, aclfromtext – convert internal representation to or from external representation												
SYNOPSIS	<pre>cc [flag...] file... -lsec [library...] #include <sys/acl.h> char *acltotext(aclent_t *aclbufp, int aclcnt); aclent_t *aclfromtext(char *acltextp, int *aclcnt);</pre>												
DESCRIPTION	The <code>acltotext()</code> function converts an internal ACL representation pointed to by <code>aclbufp</code> into an external ACL representation. The space for the external text string is obtained using <code>malloc(3C)</code>. The caller is responsible for freeing the space upon completion.. The <code>aclfromtext()</code> function converts an external ACL representation pointed to by <code>acltextp</code> into an internal ACL representation. The space for the list of ACL entries is obtained using <code>malloc(3C)</code>. The caller is responsible for freeing the space upon completion. The <code>aclcnt</code> argument indicates the number of ACL entries found. An external ACL representation is defined as follows: <pre><acl_entry>[,<acl_entry>]...</pre> Each <code><acl_entry></code> contains one ACL entry. The external representation of an ACL entry contains two or three colon-separated fields. The first field contains the ACL entry tag type. The entry type keywords are defined as: <table> <tr> <td>user</td><td>This ACL entry with no UID specified in the ACL entry ID field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number.</td></tr> <tr> <td>group</td><td>This ACL entry with no GID specified in the ACL entry ID field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number.</td></tr> <tr> <td>other</td><td>This ACL entry specifies the access granted to any user or group that does not match any other ACL entry.</td></tr> <tr> <td>mask</td><td>This ACL entry specifies the maximum access granted to user or group entries.</td></tr> <tr> <td>default:user</td><td>This ACL entry with no uid specified in the ACL entry ID field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-ID number.</td></tr> <tr> <td>default:group</td><td>This ACL entry with no gid specified in the ACL entry ID field specifies the default access granted to the owning group of the</td></tr> </table>	user	This ACL entry with no UID specified in the ACL entry ID field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number.	group	This ACL entry with no GID specified in the ACL entry ID field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number.	other	This ACL entry specifies the access granted to any user or group that does not match any other ACL entry.	mask	This ACL entry specifies the maximum access granted to user or group entries.	default:user	This ACL entry with no uid specified in the ACL entry ID field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-ID number.	default:group	This ACL entry with no gid specified in the ACL entry ID field specifies the default access granted to the owning group of the
user	This ACL entry with no UID specified in the ACL entry ID field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number.												
group	This ACL entry with no GID specified in the ACL entry ID field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number.												
other	This ACL entry specifies the access granted to any user or group that does not match any other ACL entry.												
mask	This ACL entry specifies the maximum access granted to user or group entries.												
default:user	This ACL entry with no uid specified in the ACL entry ID field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-ID number.												
default:group	This ACL entry with no gid specified in the ACL entry ID field specifies the default access granted to the owning group of the												

acltotext(3SEC)

object. Otherwise, this ACL entry specifies the default access granted to a specific group-name or group-ID number.

`default:other` This ACL entry specifies the default access for other entry.

`default:mask` This ACL entry specifies the default access for mask entry.

The second field contains the ACL entry ID, as follows:

`uid` This field specifies a user-name, or user-ID if there is no user-name associated with the user-ID number.

`gid` This field specifies a group-name, or group-ID if there is no group-name associated with the group-ID number.

`empty` This field is used by the user and group ACL entry types.

The third field contains the following symbolic discretionary access permissions:

`r` read permission

`w` write permission

`x` execute/search permission

`—` no access

RETURN VALUES Upon successful completion, the `acltotext()` function returns a pointer to a text string. Otherwise, it returns `NULL`.

Upon successful completion, the `aclfromtext()` function returns a pointer to a list of ACL entries. Otherwise, it returns `NULL`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Unsafe

SEE ALSO `acl(2)`, `malloc(3C)`, `attributes(5)`

acos(3M)

NAME	acos – arc cosine function				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double acos(double x);</pre>				
DESCRIPTION	The <code>acos()</code> function computes the principal value of the arc cosine of x . The value of x should be in the range $[-1,1]$.				
RETURN VALUES	Upon successful completion, <code>acos()</code> returns the arc cosine of x, in the range $[0,\pi]$ radians. If the value of x is not in the range $[-1,1]$, and is not $\pm\text{Inf}$ or NaN, either 0.0 or NaN is returned and <code>errno</code> is set to EDOM. If x is NaN, NaN is returned. If x is $\pm\text{Inf}$, either 0.0 is returned and <code>errno</code> is set to EDOM, or NaN is returned and <code>errno</code> may be set to EDOM. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>acos()</code> function will fail if: EDOM The value x is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$. The <code>acos()</code> function may fail if: EDOM The value x is $\pm\text{Inf}$.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>acos()</code> . If <code>errno</code> is non-zero on return, or the value NaN is returned, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>cos(3M)</code> , <code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

NAME	acosh, asinh, atanh – inverse hyperbolic functions				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double acosh(double x); double asinh(double x); double atanh(double x);</pre>				
DESCRIPTION	The <code>acosh()</code> , <code>asinh()</code> and <code>atanh()</code> functions compute the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.				
RETURN VALUES	The <code>acosh()</code>, <code>asinh()</code> and <code>atanh()</code> functions return the inverse hyperbolic cosine, sine, and tangent of their argument, respectively. The <code>acosh()</code> function returns NaN and sets <code>errno</code> to <code>EDOM</code> when its argument is less than 1.0. The <code>atanh()</code> function returns NaN and sets <code>errno</code> to <code>EDOM</code> when its argument has absolute value greater than 1.0. The <code>atanh()</code> function returns $\pm\text{Inf}$ and sets <code>errno</code> to <code>ERANGE</code> when its argument is ± 1.0. If x is NaN, the <code>asinh()</code>, <code>acosh()</code> and <code>atanh()</code> functions return NaN. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>acosh()</code> function will fail if: EDOM The x argument is less than 1.0. The <code>atanh()</code> function will fail if: EDOM The x argument has an absolute value greater than 1.0. ERANGE The x argument has an absolute value equal to 1.0				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>cosh(3M)</code> , <code>matherr(3M)</code> , <code>sinh(3M)</code> , <code>tanh(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

asin(3M)

NAME	asin – arc sine function				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double asin(double x);</pre>				
DESCRIPTION	The <code>asin()</code> function computes the principal value of the arc sine of x . The value of x should be in the range $[-1,1]$.				
RETURN VALUES	Upon successful completion, <code>asin()</code> returns the arc sine of x, in the range $[-\pi/2, \pi/2]$ radians. If the value of x is not in the range $[-1,1]$ and is not $\pm\text{Inf}$ or NaN, either 0.0 or NaN is returned and <code>errno</code> is set to <code>EDOM</code>. If x is NaN, NaN is returned. If x is $\pm\text{Inf}$, either 0.0 is returned and <code>errno</code> is set to <code>EDOM</code> or NaN is returned and <code>errno</code> may be set to <code>EDOM</code>. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>asin()</code> function will fail if: <table><tr><td>EDOM</td><td>The value x is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$.</td></tr></table> The <code>asin()</code> function may fail if: <table><tr><td>EDOM</td><td>The value of x is $\pm\text{Inf}$.</td></tr></table>	EDOM	The value x is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$.	EDOM	The value of x is $\pm\text{Inf}$.
EDOM	The value x is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$.				
EDOM	The value of x is $\pm\text{Inf}$.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0, then call <code>asin()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>sin(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

NAME	atan2 – arc tangent function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double atan2(double <i>y</i>, double <i>x</i>);</pre>				
DESCRIPTION	The <code>atan2()</code> function computes the principal value of the arc tangent of y/x , using the signs of both arguments to determine the quadrant of the return value.				
RETURN VALUES	Upon successful completion, <code>atan2()</code> returns the arc tangent of y/x in the range $[-\pi, \pi]$ radians. If both arguments are 0.0, 0.0 is returned and <code>errno</code> may be set to <code>EDOM</code>. If x or y is NaN, NaN is returned. In IEEE 754 mode <code>atan2()</code> handles the following exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. <code>atan2(± 0, x)</code> returns ± 0 for $x > 0$ or $x = +0$; <code>atan2(± 0, x)</code> returns $\pm \pi$ for $x < 0$ or $x = -0$; <code>atan2(y, ± 0)</code> returns $\pi/2$ for $y > 0$; <code>atan2(y, ± 0)</code> returns $-\pi/2$ for $y < 0$; <code>atan2($\pm y$, Inf)</code> returns ± 0 for finite $y > 0$; <code>atan2($\pm \text{Inf}$, x)</code> returns $\pm \pi/2$ for finite x; <code>atan2($\pm y$, $-\text{Inf}$)</code> returns $\pm \pi$ for finite $y > 0$; <code>atan2($\pm \text{Inf}$, Inf)</code> returns $\pm \pi/4$; <code>atan2($\pm \text{Inf}$, $-\text{Inf}$)</code> returns $\pm 3\pi/4$. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>atan2()</code> function may fail if: <code>EDOM</code> Both arguments are 0.0.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>atan2()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>atan(3M)</code> , <code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>tan(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

atan(3M)

NAME	atan – arc tangent function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double atan(double <i>x</i>) ;</pre>				
DESCRIPTION	The <code>atan()</code> function computes the principal value of the arc tangent of x .				
RETURN VALUES	Upon successful completion, <code>atan()</code> returns the arc tangent of x in the range $[-\pi/2, \pi/2]$ radians. If x is NaN, NaN is returned. If x is $\pm\text{Inf}$, $\pm\pi/2$ is returned.				
ERRORS	No errors will occur.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>atan2(3M)</code> , <code>isnan(3M)</code> , <code>tan(3M)</code> , <code>attributes(5)</code>				

NAME	au_open, au_close, au_write – construct and write audit records				
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> int au_close(int d, int keep, short event); int au_open(void); int au_write(int d, token_t *m);</pre>				
DESCRIPTION	The <code>au_open()</code> function returns an audit record descriptor to which audit tokens can be written using <code>au_write()</code>. The audit record descriptor is an integer value that identifies a storage area where audit records are accumulated. The <code>au_close()</code> function terminates the life of an audit record <i>d</i> of type <i>event</i> started by <code>au_open()</code>. If the <i>keep</i> parameter is <code>AU_TO_NO_WRITE</code>, the data contained therein is discarded. If the <i>keep</i> parameter is <code>AU_TO_WRITE</code>, the additional parameters are used to create a header token. Depending on the audit policy information obtained by <code>auditon(2)</code>, additional tokens such as <i>sequence</i> and <i>trailer</i> tokens can be added to the record. The <code>au_close()</code> function then writes the record to the audit trail by calling <code>audit(2)</code>. Any memory used is freed by calling <code>free(3C)</code>. The <code>au_write()</code> function adds the audit token pointed to by <i>m</i> to the audit record identified by the descriptor <i>d</i>. After this call is made the audit token is no longer available to the caller.				
RETURN VALUES	Upon successful completion, <code>au_open()</code> returns an audit record descriptor. If a descriptor could not be allocated, <code>au_open()</code> returns <code>-1</code> and sets <code>errno</code> to indicate the error. Upon successful completion, <code>au_close()</code> returns 0. If <i>d</i> is an invalid or corrupted descriptor or if <code>audit()</code> fails, <code>au_close()</code> returns <code>-1</code> without setting <code>errno</code>. If <code>audit()</code> fails, <code>errno</code> is set to one of the error values described on the <code>audit(2)</code> manual page. Upon successful completion, <code>au_write()</code> returns 0. If <i>d</i> is an invalid descriptor or <i>m</i> is an invalid token, or if <code>audit()</code> fails, <code>au_write()</code> returns <code>-1</code> without setting <code>errno</code>. If <code>audit()</code> fails, <code>errno</code> is set to one of the error values described on the <code>audit(2)</code> manual page.				
ERRORS	The <code>au_open()</code> function will fail if: <table> <tr> <td><code>ENOMEM</code></td><td>The physical limits of the system have been exceeded such that sufficient memory cannot be allocated.</td></tr> <tr> <td><code>EAGAIN</code></td><td>There is currently insufficient memory available. The application can try again later.</td></tr> </table>	<code>ENOMEM</code>	The physical limits of the system have been exceeded such that sufficient memory cannot be allocated.	<code>EAGAIN</code>	There is currently insufficient memory available. The application can try again later.
<code>ENOMEM</code>	The physical limits of the system have been exceeded such that sufficient memory cannot be allocated.				
<code>EAGAIN</code>	There is currently insufficient memory available. The application can try again later.				

au_open(3BSM)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO bsmconv(1M), audit(2), auditon(2), au_preselect(3BSM), au_to(3BSM), free(3C), attributes(5)

NOTES The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.

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

au_preselect(3BSM)

/etc/security/audit_event maps audit even number to audit event names and associates

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO bsmconv(1M), getaudit(2), au_open(3BSM), getauclassent(3BSM), getauevent(3BSM), malloc(3C), audit_class(4), audit_event(4), attributes(5)

NOTES au_preselect () is normally called prior to constructing and writing an audit record. If the event is not preselected, the overhead of constructing and writing the record can be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.

NAME	au_to, au_to_arg, au_to_arg32, au_to_arg64, au_to_attr, au_to_cmd, au_to_data, au_to_groups, au_to_in_addr, au_to_ipc, au_to_important, au_to_me, au_to_newgroups, au_to_opaque, au_to_path, au_to_process, au_to_process_ex, au_to_return, au_to_return32, au_to_return64, au_to_socket, au_to_subject, au_to_subject_ex, au_to_text – create audit record tokens
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/types.h> #include <sys/vnode.h> #include <netinet/in.h> #include <bsm/libbsm.h> token_t *au_to_arg(char n, char *text, uint32_t v); token_t *au_to_arg32(char n, char *text, uint32_t v); token_t *au_to_arg64(char n, char *text, uint64_t v); token_t *au_to_attr(struct vattr *attr); token_t *au_to_cmd(uint_t argc, char **argv, char **envp); token_t *au_to_data(char unit_print, char unit_type, char unit_count, char *p); token_t *au_to_groups(int *groups); token_t *au_to_in_addr(struct in_addr *internet_addr); token_t *au_to_ipc(char type, int id); token_t *au_to_important(u_short_t ipp); token_t *au_to_me(void); token_t *au_to_newgroups(int n, gid_t *groups); token_t *au_to_opaque(char *data, short bytes); token_t *au_to_path(char *path); token_t *au_to_process(auid_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_t *tid); token_t *au_to_process_ex(auid_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_addr_t *tid); token_t *au_to_return(char number, uin32_t value); token_t *au_to_return32(char number, uin32_t value); token_t *au_to_return64(char number, uin64_t value); token_t *au_to_socket(struct oldsocket *so); token_t *au_to_subject(auid_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_t *tid);</pre>

au_to(3BSM)

DESCRIPTION

```
token_t *au_to_subject_ex(au_id_t auid, uid_t euid, gid_t egid, uid_t  
    ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_addr_t *tid);
```

```
token_t *au_to_text(char *text);
```

The `au_to_arg()`, `au_to_arg32()`, and `au_to_arg64()` functions format the data in *v* into an “argument token”. The *n* argument indicates the argument number. The *text* argument is a null-terminated string describing the argument.

The `au_to_attr()` function formats the data pointed to by *attr* into a “vnode attribute token”.

The `au_to_cmd()` function formats the data pointed to by *argv* into a “command token”. A command token reflects a command and its parameters as entered. For example, the `pfexec(1)` utility uses `au_to_cmd()` to record the command and arguments it reads from the command line.

The `au_to_data()` function formats the data pointed to by *p* into an “arbitrary data token”. The *unit_print* parameter determines the preferred display base of the data and is one of `AUP_BINARY`, `AUP_OCTAL`, `AUP_DECIMAL`, `AUP_HEX`, or `AUP_STRING`. The *unit_type* parameter defines the basic unit of data and is one of `AUR_BYTE`, `AUR_CHAR`, `AUR_SHORT`, `AUR_INT`, or `AUR_LONG`. The *unit_count* parameter specifies the number of basic data units to be used and must be positive.

The `au_to_groups()` function formats the array of 16 integers pointed to by *groups* into a “groups token”. The `au_to_newgroups()` function (see below) should be used in place of this function.

The `au_to_in_addr()` function formats the data pointed to by *internet_addr* into an “internet address token”.

The `au_to_ipc()` function formats the data in the *id* parameter into an “interprocess communications ID token”.

The `au_to_iport()` function formats the data pointed to by *iport* into an “ip port address token”.

The `au_to_me()` function collects audit information from the current process and creates a “subject token” by calling `au_to_subject()`.

The `au_to_newgroups()` function formats the array of *n* integers pointed to by *groups* into a “newgroups token”. This function should be used in place of `au_to_groups()`.

The `au_to_opaque()` function formats the *bytes* bytes pointed to by *data* into an “opaque token”. The value of *size* must be positive.

The `au_to_path()` function formats the path name pointed to by *path* into a “path token.”

The `au_to_process()` function formats an *auuid* (audit user ID), an *euuid* (effective user ID), an *egid* (effective group ID), a *ruuid* (real user ID), a *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), and a *tid* (audit terminal ID containing an IPv4 IP address), into a “process token”. A process token should be used when the process is the object of an action (ie. when the process is the receiver of a signal). The `au_to_process_ex()` function (see below) should be used in place of this function.

The `au_to_process_ex()` function formats an *auuid* (audit user ID), an *euuid* (effective user ID), an *egid* (effective group ID), a *ruuid* (real user ID), a *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), and a *tid* (audit terminal ID containing an IPv4 or IPv6 IP address), into a “process token”. A process token should be used when the process is the object of an action (that is, when the process is the receiver of a signal). This function should be used in place of `au_to_process()`.

The `au_to_return()`, `au_to_return32()`, and `au_to_return64()` functions format an error number *number* and a return value *value* into a “return value token”.

The `au_to_socket()` function format the data pointed to by *so* into a “socket token.”

The `au_to_subject()` function formats an *auuid* (audit user ID), an *euuid* (effective user ID), an *egid* (effective group ID), a *ruuid* (real user ID), an *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), an *tid* (audit terminal ID containing an IPv4 IP address), into a “subject token”. The `au_to_subject_ex()` function (see below) should be used in place of this function.

The `au_to_subject_ex()` function formats an *auuid* (audit user ID), an *euuid* (effective user ID), an *egid* (effective group ID), a *ruuid* (real user ID), an *rgid* (real group ID), a *pid* (process ID), an *sid* (audit session ID), an *tid* (audit terminal ID containing an IPv4 or IPv6 IP address), into a “subject token”. This function should be used in place of `au_to_subject()`.

The `au_to_text()` function formats the null-terminated string pointed to by *text* into a “text token”.

RETURN VALUES

These functions return `NULL` if memory cannot be allocated to put the resultant token into, or if an error in the input is detected.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO

`bsmconv(1M)`, `au_open(3BSM)`, `attributes(5)`

au_to(3BSM)

NOTES	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See <code>bsmconv(1M)</code> for more information.

NAME	au_user_mask – get user’s binary preselection mask							
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> int au_user_mask(char *username, au_mask_t *mask_p);</pre>							
DESCRIPTION	au_user_mask() reads the default, system wide audit classes from audit_control(4), combines them with the per-user audit classes from the audit_user(4) database, and updates the binary preselection mask pointed to by mask_p with the combined value. The audit flags in the flags field of the audit_control(4) database and the always-audit-flags and never-audit-flags from the audit_user(4) database represent binary audit classes. These fields are combined by au_preselect(3BSM) as follows: $\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$ au_user_mask() only fails if both the both the audit_control(4) and the audit_user(4) database entries could not be retrieved. This allows for flexible configurations.							
RETURN VALUES	au_user_mask() returns: 0 Success. –1 Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.							
FILES	/etc/security/audit_control	contains default parameters read by the audit daemon, auditd(1M)						
	/etc/security/audit_user	stores per-user audit event mask						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Interface Stability	Stable							
MT-Level	MT-Safe							
SEE ALSO	login(1), bsmconv(1M), getaudit(2), setaudit(2), au_preselect(3BSM), getacinfo(3BSM), getauusernam(3BSM), audit_control(4), audit_user(4), attributes(5)							
NOTES	au_user_mask() should be called by programs like login(1) which set a process’s preselection mask with setaudit(2). getaudit(2) should be used to obtain audit characteristics for the current process.							

`au_user_mask(3BSM)`

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

NAME	bgets – read stream up to next delimiter
SYNOPSIS	<pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> char *bgets(char *buffer, size_t count, FILE *stream, const char *breakstring);</pre>
DESCRIPTION	The <code>bgets()</code> function reads characters from <i>stream</i> into <i>buffer</i> until either <i>count</i> is exhausted or one of the characters in <i>breakstring</i> is encountered in the stream. The read data is terminated with a null byte ('\0') and a pointer to the trailing null is returned. If a <i>breakstring</i> character is encountered, the last non-null is the delimiter character that terminated the scan. Note that, except for the fact that the returned value points to the end of the read string rather than to the beginning, the call <pre>bgets(buffer, sizeof buffer, stream, "\n");</pre> is identical to <pre>fgetc (buffer, sizeof buffer, stream);</pre> There is always enough room reserved in the buffer for the trailing null character. If <i>breakstring</i> is a null pointer, the value of <i>breakstring</i> from the previous call is used. If <i>breakstring</i> is null at the first call, no characters will be used to delimit the string.
RETURN VALUES	NULL is returned on error or end-of-file. Reporting the condition is delayed to the next call if any characters were read but not yet returned.
EXAMPLES	EXAMPLE 1 Example of the <code>bgets()</code> function. The following example prints the name of the first user encountered in <code>/etc/passwd</code>, including a trailing ":" <pre>#include <stdio.h> #include<libgen.h> int main() { char buffer[8]; FILE *fp; if ((fp = fopen("/etc/passwd","r")) == NULL) { perror("/etc/passwd"); return 1; } if (bgets(buffer, 8, fp, ":") == NULL) { perror("bgets"); return 1; } (void) puts(buffer); return 0; }</pre>

bgets(3GEN)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO gets(3C), attributes(5)

NOTES When compiling multithread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

NAME	bufsplit – split buffer into fields				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> size_t bufsplit(char *<i>buf</i>, size_t <i>n</i>, char **<i>a</i>);</pre>				
DESCRIPTION	bufsplit() examines the buffer, <i>buf</i>, and assigns values to the pointer array, <i>a</i>, so that the pointers point to the first <i>n</i> fields in <i>buf</i> that are delimited by TABs or NEWLINEs. To change the characters used to separate fields, call bsplit() with <i>buf</i> pointing to the string of characters, and <i>n</i> and <i>a</i> set to zero. For example, to use colon (:), period (.), and comma (,), as separators along with TAB and NEWLINE: <pre>bufsplit (":.,\t\n", 0, (char**)0);</pre>				
RETURN VALUES	The number of fields assigned in the array <i>a</i> . If <i>buf</i> is zero, the return value is zero and the array is unchanged. Otherwise the value is at least one. The remainder of the elements in the array are assigned the address of the null byte at the end of the buffer.				
EXAMPLES	EXAMPLE 1 Example of bsplit() function. <pre>/* * set a[0] = "This", a[1] = "is", a[2] = "a", * a[3] = "test" */ bufsplit("This\tis\ta\ttest\n", 4, a);</pre>				
NOTES	bufsplit() changes the delimiters to null bytes in <i>buf</i>. When compiling multithreaded applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multithreaded applications.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	attributes(5)				

cbrt(3M)

NAME	cbt – cube root function
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double cbt(double <i>x</i>);</pre>
DESCRIPTION	The <code>cbt()</code> function computes the cube root of <i>x</i> .
RETURN VALUES	On successful completion, <code>cbt()</code> returns the cube root of <i>x</i> . If <i>x</i> is NaN, <code>cbt()</code> returns NaN.
ERRORS	No errors will occur.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `attributes(5)`

NAME	ceil – ceiling value function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double ceil(double <i>x</i>);</pre>				
DESCRIPTION	The <code>ceil()</code> function computes the smallest integral value not less than <i>x</i> .				
RETURN VALUES	Upon successful completion, <code>ceil()</code> returns the smallest integral value not less than <i>x</i>, expressed as a type <code>double</code>. If <i>x</i> is NaN, NaN is returned. If <i>x</i> is $\pm\text{Inf}$ or ± 0, <i>x</i> is returned.				
ERRORS	No errors will occur.				
USAGE	The integral value returned by <code>ceil()</code> as a <code>double</code> may not be expressible as an <code>int</code> or <code>long int</code> . The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>floor(3M)</code> , <code>isnan(3M)</code> , <code>attributes(5)</code>				

config_admin(3CFGADM)

NAME	config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – configuration administration interface
SYNOPSIS	<pre>cc [flag] file -lcfgadm [library...] #include <config_admin.h> cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int num_ap_ids, char * const *ap_ids, const char *options, struct cfga_confirm *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags); cfga_err_t config_private_func(const char *function, int num_ap_ids, char * const *ap_ids, const char *options, struct cfga_confirm *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags); cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const char *options, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags); cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids, struct cfga_list_data **ap_id_list, int *nlist, const char *options, const char *listops, char **errstring, cfga_flags_t flags); int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t ap_id2); void config_unload_libs(void); const char *config_strerror(cfga_err_t cfgerrnum);</pre>
Deprecated Interfaces	The following interfaces have been deprecated and their use is strongly discouraged: <pre>cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct cfga_stat_data *buf, const char *options, char **errstring); cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist, const char *options, char **errstring);</pre>
HARDWARE DEPENDENT LIBRARY SYNOPSIS	The config_admin library is a generic interface that is used for dynamic configuration, (DR). Each piece of hardware that supports DR must supply a hardware-specific <i>plugin</i> library that contains the entry points listed in this subsection. The generic library will locate and link to the appropriate library to effect DR operations. The interfaces specified in this subsection are really "hidden" from users of the generic libraries. It is, however, necessary that writers of the hardware-specific plug in libraries know what these interfaces are. <pre>cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char *ap_id, const char *options, struct cfga_confirm *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags); cfga_err_t cfga_private_func(const char *function, const char *ap_id, const char *options, struct cfga_confirm *confp, struct cfga_msg</pre>


```

    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);

```

Deprecated Interfaces

The following interfaces have been deprecated and their use is strongly discouraged:

```

cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data
    **ap_id_list, int *nlist, const char *options, char **errstring);

```

DESCRIPTION

The `config_*`() functions provide a hardware independent interface to hardware-specific system configuration administration functions. The `cfga_*`() functions are provided by hardware-specific libraries that are dynamically loaded to handle configuration administration functions in a hardware-specific manner.

The `libcfigadm` library is used to provide the services of the `cfgadm(1M)` command. The hardware-specific libraries are located in `/usr/platform/${machine}/lib/cfigadm`, `/usr/platform/${arch}/lib/cfigadm`, and `/usr/lib/cfigadm`. The hardware-specific library names are derived from the driver name or from class names in device tree nodes that identify attachment points.

The `config_change_state()` function performs operations that change the state of the system configuration. The `state_change_cmd` argument can be one of the following: `CFG_CMD_INSERT`, `CFG_CMD_REMOVE`, `CFG_CMD_DISCONNECT`, `CFG_CMD_CONNECT`, `CFG_CMD_CONFIGURE`, or `CFG_CMD_UNCONFIGURE`. The `state_change_cmd` `CFG_CMD_INSERT` is used to prepare for manual insertion or to activate automatic hardware insertion of an occupant. The `state_change_cmd` `CFG_CMD_REMOVE` is used to prepare for manual removal or activate automatic hardware removal of an occupant. The `state_change_cmd` `CFG_CMD_DISCONNECT` is used to disable normal communication to or from an occupant in a receptacle. The `state_change_cmd` `CFG_CMD_CONNECT` is used to enable communication to or from an occupant in a receptacle. The `state_change_cmd` `CFG_CMD_CONFIGURE` is used to bring the hardware resources contained on, or attached to, an occupant into the realm of Solaris, allowing use of the occupant's hardware resources by the system. The `state_change_cmd` `CFG_CMD_UNCONFIGURE` is used to remove the hardware resources contained on, or attached to, an occupant from the realm of Solaris, disallowing further use of the occupant's hardware resources by the system.

config_admin(3CFGADM)

The *flags* argument may contain one or both of the defined flags, `CFGA_FLAG_FORCE` and `CFGA_FLAG_VERBOSE`. If the `CFGA_FLAG_FORCE` flag is asserted certain safety checks will be overridden. For example, this may not allow an occupant in the failed condition to be configured, but might allow an occupant in the failing condition to be configured. Acceptance of a force is hardware dependent. If the `CFGA_FLAG_VERBOSE` flag is asserted hardware-specific details relating to the operation are output utilizing the `cfga_msg` mechanism.

The `config_private_func()` function invokes private hardware-specific functions.

The `config_test()` function is used to initiate testing of the specified attachment point.

The *num_ap_ids* argument specifies the number of *ap_ids* in the *ap_ids* array. The *ap_ids* argument points to an array of *ap_ids*.

The *ap_id* argument points to a single *ap_id*.

The *function* and *options* strings conform to the `getsubopt(3C)` syntax convention and are used to supply hardware-specific function or option information. No generic hardware-independent functions or options are defined.

The `cfga_confirm` structure referenced by *confp* provides a call-back interface to get permission to proceed should the requested operation require, for example, a noticeable service interruption. The `cfga_confirm` structure includes the following members:

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

The `confirm()` function is called with two arguments: the generic pointer *appdata_ptr* and the message detailing what requires confirmation. The generic pointer *appdata_ptr* is set to the value passed in in the `cfga_confirm` structure member `appdata_ptr` and can be used in a graphical user interface to relate the `confirm` function call to the `config_*` call. The `confirm` function should return 1 to allow the operation to proceed and 0 otherwise.

The `cfga_msg` structure referenced by *msgp* provides a call-back interface to output messages from a hardware-specific library. In the presence of the `CFGA_FLAG_VERBOSE` flag, these messages can be informational; otherwise they are restricted to error messages. The `cfga_msg` structure includes the following members:

```
void  (*message_routine)(void *appdata_ptr, const char *message);
void  *appdata_ptr;
```

The `message_routine()` function is called with two arguments: the generic pointer *appdata_ptr* and the message. The generic pointer *appdata_ptr* is set to the value passed in in the `cfga_confirm` structure member `appdata_ptr` and can be used in a graphical user interface to relate the `message_routine()` function call to the `config_*` call. The messages must be in the native language specified by the `LC_MESSAGES` locale category; see `setlocale(3C)`.

For some generic errors a hardware-specific error message can be returned. The storage for the error message string, including the terminating null character, is allocated by the `config_*` functions using `malloc(3C)` and a pointer to this storage returned through `errstring`. If `errstring` is NULL no error message will be generated or returned. If `errstring` is not NULL and no error message is generated, the pointer referenced by `errstring` will be set to NULL. It is the responsibility of the function calling `config_*` to deallocate the returned storage using `free(3C)`. The error messages must be in the native language specified by the `LC_MESSAGES` locale category; see `setlocale(3C)`.

The `config_list_ext()` function provides the listing interface. When supplied with a list of `ap_ids` through the first two arguments, it returns an array of `cfga_list_data_t` structures for each attachment point specified. If the first two arguments are 0 and NULL respectively, then all attachment points in the device tree will be listed. Additionally, dynamic expansion of an attachment point to list dynamic attachment points may also be requested by passing the `CFG_FLAG_LIST_ALL` flag through the `flags` argument. Storage for the returned array of `stat` structures is allocated by the `config_list_ext()` function using `malloc(3C)`. This storage must be freed by the caller of `config_list_ext()` by using `free(3C)`.

The `cfga_list_data` structure includes the following members:

```

cfga_log_ext_t      ap_log_id;          /* Attachment point logical id */
cfga_phys_ext_t     ap_phys_id;         /* Attachment point physical id */
cfga_class_t        ap_class;           /* Attachment point class */
cfga_stat_t         ap_r_state;         /* Receptacle state */
cfga_stat_t         ap_o_state;         /* Occupant state */
cfga_cond_t         ap_cond;            /* Attachment point condition */
cfga_busy_t         ap_busy;            /* Busy indicator */
time_t              ap_status_time;     /* Attachment point last change */
cfga_info_t         ap_info;            /* Miscellaneous information */
cfga_type_t         ap_type;            /* Occupant type */

```

The types are defined as follows:

```

typedef char cfga_log_ext_t[CFG_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFG_PHYS_EXT_LEN];
typedef char cfga_class_t[CFG_CLASS_LEN];
typedef char cfga_info_t[CFG_INFO_LEN];
typedef char cfga_type_t[CFG_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;

```

The `listopts` argument to `config_list_ext()` conforms to the `getsubopt(3C)` syntax and is used to pass listing sub-options. Currently, only the sub-option `class=class_name` is supported. This list option restricts the listing to attachment points of class `class_name`.

config_admin(3CFGADM)

The *listopts* argument to `cfga_list_ext()` is reserved for future use.

Hardware-specific libraries should ignore this argument if it is NULL. If *listopts* is not NULL and is not supported by the hardware-specific library, an appropriate error code should be returned.

The `ap_log_id` and the `ap_phys_id` members give the hardware-specific logical and physical names of the attachment point. The `ap_busy` member indicates activity is present that may result in changes to state or condition. The `ap_status_time` member provides the time at which either the `ap_r_state`, `ap_o_state`, or `ap_cond` field of the attachment point last changed. The `ap_info` member is available for the hardware-specific code to provide additional information about the attachment point. The `ap_class` member contains the attachment point class (if any) for an attachment point. The `ap_class` member is filled in by the generic library. If the `ap_log_id` and `ap_phys_id` members are not filled in by the hardware-specific library, the generic library will fill in these members using a generic format. The remaining members are the responsibility of the corresponding hardware-specific library.

The `ap_log_id`, `ap_phys_id`, `ap_info`, `ap_class`, and `ap_type` members are fixed-length strings. If the actual string is shorter than the size of the member, it will be null-terminated. Because of this, programs should not rely on there being a terminating null character. When printing these fields, the following format is suggested:

```
printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);
```

The `config_stat()`, `config_list()`, `cfga_stat()`, and `cfga_list()` functions and the `cfga_stat_data` data structure are deprecated interfaces and are provided solely for backward compatibility. Use of these interfaces is strongly discouraged.

The `config_ap_id_cmp` function performs a hardware dependent comparison on two *ap_ids*, returning an equal to, less than or greater than indication in the manner of `strcmp(3C)`. Each argument is either a `cfga_ap_id_t` or can be a null-terminated string. This function can be used when sorting lists of *ap_ids*, for example with `qsort(3C)`, or when selecting entries from the result of a `config_list` function call.

The `config_unload_libs` function unlinks all previously loaded hardware-specific libraries.

The `config_strerror` function can be used to map an error return value to an error message string. See RETURN VALUES. The returned string should not be overwritten. `config_strerror` returns NULL if *cfgerrnum* is out-of-range.

The `cfga_help` function can be used request that a hardware-specific library output it's localized help message.

RETURN VALUES	The <code>config_*</code> () and <code>cfga_*</code> () functions return the following values. Additional error information may be returned through <i>errstring</i> if the return code is not <code>CFGA_OK</code> . See DESCRIPTION for details.
<code>CFGA_BUSY</code>	The command was not completed due to an element of the system configuration administration system being busy.
<code>CFGA_ATTR_INVALID</code>	No attachment points with the specified attributes exists
<code>CFGA_ERROR</code>	An error occurred during the processing of the requested operation. This error code includes validation of the command arguments by the hardware-specific code.
<code>CFGA_INSUFFICIENT_CONDITION</code>	Operation failed due to attachment point condition.
<code>CFGA_INVALID</code>	The system configuration administration operation requested is not supported on the specified attachment point.
<code>CFGA_LIB_ERROR</code>	A procedural error occurred in the library, including failure to obtain process resources such as memory and file descriptors.
<code>CFGA_NACK</code>	The command was not completed due to a negative acknowledgement from the <i>confp</i> -> <i>confirm</i> function.
<code>CFGA_NO_LIB</code>	A hardware-specific library could not be located using the supplied <i>ap_id</i> .
<code>CFGA_NOTSUPP</code>	System configuration administration is not supported on the specified attachment point.
<code>CFGA_OK</code>	The command completed as requested.
<code>CFGA_OPNOTSUPP</code>	System configuration administration operation is not supported on this attachment point.
<code>CFGA_PRIV</code>	The caller does not have the required process privileges. For example, if configuration administration is performed through a device driver, the permissions on the device node would be used to control access.
<code>CFGA_SYSTEM_BUSY</code>	The command required a service interruption and was not completed due to

config_admin(3CFGADM)

a part of the system that could not be quiesced.

ERRORS Many of the errors returned by the system configuration administration functions are hardware-specific. The strings returned in *errstring* may include the following:

attachment point *ap_id* not known

The attachment point detailed in the error message does not exist.

unknown hardware option *option* for *operation*

An unknown option was encountered in the *options* string.

hardware option *option* requires a value

An option in the *options* string should have been of the form *option=value*.

listing option *list_option* requires a value

An option in the listopts string should have been of the form *option=value*.

hardware option *option* does not require a value

An option in the *options* string should have been a simple option.

attachment point *ap_id* is not configured

A *config_change_state* command to CFGA_CMD_UNCONFIGURE an occupant was made to an attachment point whose occupant was not in the CFGA_STAT_CONFIGURED state.

attachment point *ap_id* is not unconfigured

A *config_change_state* command requiring an unconfigured occupant was made to an attachment point whose occupant was not in the CFGA_STAT_UNCONFIGURED state.

attachment point *ap_id* condition not satisfactory

A *config_change_state* command was made to an attachment point whose condition prevented the operation.

attachment point *ap_id* in condition *condition* cannot be used

A *config_change_state* operation with force indicated was directed to an attachment point whose condition fails the hardware dependent test.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

SEE ALSO cfgadm(1M), devinfo(1M), dlopen(3DL), dlsym(3DL), free(3C), getsubopt(3C), malloc(3C), qsort(3C), setlocale(3C), strcmp(3C), libcfgadm(3LIB), attributes(5)

NOTES Applications using this library should be aware that the underlying implementation may use system services which alter the contents of the external variable `errno` and may use file descriptor resources.

The following code shows the intended error processing when `config_*()` returns a value other than `CFGA_OK`:

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

Reference should be made to the Hardware Specific Guide for details of System Configuration Administration support.

ConnectToServer(3DMI)

NAME	ConnectToServer – connect to a DMI service provider				
SYNOPSIS	<pre>cc [flag ...] file ... -ldmici -ldmimi [library ...] #include <dmi/api.hh> bool_t ConnectToServer(ConnectI *argp, DmiRpcHandle *dmi_rpc_handle);</pre>				
DESCRIPTION	The <code>ConnectToServer()</code> function enables a management application or a component instrumentation to connect to a DMI service provider. The <i>argp</i> parameter is an input parameter that uses the following data structure: <pre>struct ConnectIN { char *host; const char *nettype; ServerType servertime; RpcType rpctype; }</pre> The <i>host</i> member indicates the host on which the service provider is running. The default is <i>localhost</i>. The <i>nettype</i> member specifies the type of transport RPC uses. The default is <i>netpath</i>. The <i>servertime</i> member indicates whether the connecting process is a management application or a component instrumentation. The <i>rpctype</i> member specifies the type of RPC, either ONC or DCE. Only ONC is supported in the Solaris 7 release. The <i>dmi_rpc_handle</i> parameter is the output parameter that returns DMI RPC handle.				
RETURN VALUES	The <code>ConnectToServer()</code> function returns TRUE if successful, otherwise FALSE.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-level</td><td>Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-level	Safe				
SEE ALSO	<code>DisconnectToServer(3DMI)</code> , <code>attributes(5)</code>				

NAME	copylist – copy a file into memory				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *copylist(const char *<i>filenm</i>, off_t *<i>szptr</i>);</pre>				
DESCRIPTION	The <code>copylist()</code> function copies a list of items from a file into freshly allocated memory, replacing new-lines with null characters. It expects two arguments: a pointer <i>filenm</i> to the name of the file to be copied, and a pointer <i>szptr</i> to a variable where the size of the file will be stored. Upon success, <code>copylist()</code> returns a pointer to the memory allocated. Otherwise it returns NULL if it has trouble finding the file, calling <code>malloc()</code>, or reading the file.				
USAGE	The <code>copylist()</code> function has a transitional interface for 64-bit file offsets. See <code>lf64(5)</code> .				
EXAMPLES	EXAMPLE 1 Example of <code>copylist()</code> function. <pre>/* read "file" into buf */ off_t size; char *buf; buf = copylist("file", &size); if (buf) { for (i=0; i<size; i++) if (buf[i]) putchar(buf[i]); else putchar('\n'); } } else { fprintf(stderr, "%s: Copy failed for "file".\n", argv[0]); exit (1); }</pre>				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>malloc(3C)</code> , <code>attributes(5)</code> , <code>lf64(5)</code>				
NOTES	When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multithreaded applications.				

copysign(3M)

NAME	copysign – return magnitude of first argument and sign of second argument
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double copysign(double <i>x</i>, double <i>y</i>);</pre>
DESCRIPTION	The copysign() function returns a value with the magnitude of <i>x</i> and the sign of <i>y</i> . It produces a NaN with the sign of <i>y</i> if <i>x</i> is a NaN.
RETURN VALUES	The copysign() function returns a value with the magnitude of <i>x</i> and the sign of <i>y</i> .
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO attributes(5)

NAME	cos – cosine function
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double cos(double <i>x</i>);</pre>
DESCRIPTION	The <code>cos()</code> function computes the cosine of <i>x</i> , measured in radians.
RETURN VALUES	Upon successful completion, <code>cos()</code> returns the cosine of <i>x</i> . If <i>x</i> is NaN or $\pm\text{Inf}$, NaN is returned.
ERRORS	No errors will occur.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `acos(3M)`, `isnan(3M)`, `sin(3M)`, `tan(3M)`, `attributes(5)`

cosh(3M)

NAME	cosh – hyperbolic cosine function				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double cosh(double x);</pre>				
DESCRIPTION	The <code>cosh()</code> function computes the hyperbolic cosine of x .				
RETURN VALUES	Upon successful completion, <code>cosh()</code> returns the hyperbolic cosine of x. If the result would cause an overflow, <code>HUGE_VAL</code> is returned and <code>errno</code> is set to <code>ERANGE</code>. If x is NaN, NaN is returned. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>cosh()</code> function will fail if: <code>ERANGE</code> The result would cause an overflow.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>cosh()</code> . If <code>errno</code> is non-zero on return, or the returned value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>acosh(3M)</code> , <code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>sinh(3M)</code> , <code>tanh(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

NAME	cpc – hardware performance counters										
DESCRIPTION	The UltraSPARC and Pentium microprocessor families contain <i>hardware performance counters</i> that allow the measurement of many different hardware events related to CPU behavior, including instruction and data cache misses as well as various internal states of the processor. More recent processors allow a variety of events to be captured. The counters can be configured to count user events or system events, or both. The two processor families currently share the restriction that only two event types can be measured simultaneously. UltraSPARC III and Pentium II processors are able to generate an interrupt on counter overflow, allowing the counters to be used for various forms of profiling. This manual page describes a set of APIs that allow Solaris applications to use these counters. Applications can measure their own behavior, the behavior of other applications, or the behavior of the whole system.										
Shared counters or private counters?	There are two principal models for using these performance counters. Some users of these statistics wish to observe system-wide behavior; others wish to view the performance counters as part of the register set exported by each LWP. On a machine performing more than one activity, these two models are in conflict because the counters represent a critical hardware resource that cannot simultaneously be both shared and private.										
Generic or specific events?	Although some events are common to all processors, it is apparent that the counters expose a great deal of the specific implementation details of the processor architecture. For this reason, events are specified by name using a string-based hardware event specification language. The values of the tokens in the language vary from processor model to processor model, and can only be interpreted with reference to the relevant hardware documentation. The functions provided to specify the strings use environment variables or arguments so that the names do not have to be compiled in applications, thus extending their longevity and portability across platforms and processor generations.										
Configuration Interfaces	The following configuration interfaces are provided: <table> <tr> <td>cpc_version(3CPC)</td><td>check the version the application was compiled with against the version of the library</td></tr> <tr> <td>cpc_getcpuver(3CPC)</td><td>determine the performance counter version of the current CPU</td></tr> <tr> <td>cpc_getcciname(3CPC)</td><td>return the corresponding printable string to describe that interface</td></tr> <tr> <td>cpc_getnpic(3CPC)</td><td>return the number of valid counter registers in the cpc_event(3CPC) data structure</td></tr> <tr> <td>cpc_getcpuref(3CPC)</td><td>return a reference to the corresponding processor documentation</td></tr> </table>	cpc_version(3CPC)	check the version the application was compiled with against the version of the library	cpc_getcpuver(3CPC)	determine the performance counter version of the current CPU	cpc_getcciname(3CPC)	return the corresponding printable string to describe that interface	cpc_getnpic(3CPC)	return the number of valid counter registers in the cpc_event(3CPC) data structure	cpc_getcpuref(3CPC)	return a reference to the corresponding processor documentation
cpc_version(3CPC)	check the version the application was compiled with against the version of the library										
cpc_getcpuver(3CPC)	determine the performance counter version of the current CPU										
cpc_getcciname(3CPC)	return the corresponding printable string to describe that interface										
cpc_getnpic(3CPC)	return the number of valid counter registers in the cpc_event(3CPC) data structure										
cpc_getcpuref(3CPC)	return a reference to the corresponding processor documentation										

cpc(3CPC)

Performance Counter Access	Performance counters can be present in hardware but not accessible because either some of the necessary system software components are not available or not installed, or the counters may be in use by other processes. The <code>cpc_access(3CPC)</code> function determines the accessibility of the counters and should be invoked before any attempt to program the counters.
Programming events	Events are specified using a <code>getsubopt(3C)</code> -style language for both the events and the additional control bits that determine what causes the counters to increment. The <code>cpc_strtoevent()</code> function translates a string to an event specification which can then be used to program the counters. The <code>cpc_eventtostr()</code> function returns the canonical form of the string that corresponds to a particular event. The <code>cpc_getusage(3CPC)</code> function returns a string that specifies the syntax of the string, while <code>cpc_walk_names(3CPC)</code> allows the caller to apply a function to each possible event supported on the relevant processor.
Performance counter context	Each processor on the system possesses its own set of performance counter registers. For a single process, it is often desirable to maintain the illusion that the counters are an intrinsic part of that process (whichever processors it runs on), since this allows the events to be directly attributed to the process without having to make passive all other activity on the system. To achieve this behavior, the library associates <i>performance counter context</i> with each LWP in the process; the context consists of a small amount of kernel memory to hold the counter values when the LWP is not running, and some simple kernel functions to save and restore those counter values from and to the hardware registers when the LWP performs a normal context switch. A process can only observe and manipulate its own copy of the performance counter control and data registers.
Performance Counters In Other Processes	Though applications can be modified to instrument themselves as demonstrated above, it is frequently useful to be able to examine the behavior of an existing application without changing the source code. A separate library, <code>libpctx</code>, provides a simple set of interfaces that use the facilities of <code>proc(4)</code> to control a target process, and together with functions in <code>libcpc</code>, allow <code>truss</code>-like tools to be constructed to measure the performance counters in other applications. An example of one such application is <code>cputrack(1)</code>. The functions in <code>libpctx</code> are independent of those in <code>libcpc</code>. These functions manage a process using an event-loop paradigm — that is, the execution of certain system calls by the controlled process cause the library to stop the controlled process and execute callback functions in the context of the controlling process. These handlers can perform various operations on the target process using APIs in <code>libpctx</code> and <code>libcpc</code> that consume <code>pctx_t</code> handles.

cpc(3CPC)

SEE ALSO | cputrack(1), cpustat(1M), cpc_access(3CPC), cpc_bind_event(3CPC),
cpc_count_usr_events(3CPC), cpc_pctx_bind_event(3CPC),
cpc_event(3CPC), cpc_event_diff(3CPC), cpc_getcpuver(3CPC),
cpc_seterrfn(3CPC), cpc_shared_bind_event(3CPC),
cpc_strtoevent(3CPC), cpc_version(3CPC), pctx_capture(3CPC),
pctx_set_events(3CPC), proc(4).

cpc_access(3CPC)

NAME	cpc_access – test access CPU performance counters								
SYNOPSIS	<pre>cc [flag...] file... -lcpc [library...] #include <libcpc.h> int cpc_access(void);</pre>								
DESCRIPTION	Access to CPU performance counters is possible only on systems where the appropriate hardware exists and is correctly configured. The <code>cpc_access()</code> function <i>must</i> be used to determine if the hardware exists and is accessible on the platform before any of the interfaces that use the counters are invoked. When the hardware is available, access to the per-process counters is always allowed to the process itself, and allowed to other processes mediated using the existing security mechanisms of <code>/proc</code>.								
RETURN VALUES	Upon successful completion, <code>cpc_access()</code> returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate the error. By default, two common <code>errno</code> values are decoded and cause the library to print an error message using its reporting mechanism. See <code>cpc_seterrfn(3CPC)</code> for a description of how this behavior can be modified.								
ERRORS	The <code>cpc_access()</code> function will fail if: <table><tr><td>EAGAIN</td><td>Another process may be sampling system-wide CPU statistics.</td></tr><tr><td>ENOSYS</td><td>CPU performance counters are inaccessible on this machine. This error can occur when the machine supports CPU performance counters, but some software components are missing. Check to see that all CPU Performance Counter packages have been correctly installed.</td></tr></table>	EAGAIN	Another process may be sampling system-wide CPU statistics.	ENOSYS	CPU performance counters are inaccessible on this machine. This error can occur when the machine supports CPU performance counters, but some software components are missing. Check to see that all CPU Performance Counter packages have been correctly installed.				
EAGAIN	Another process may be sampling system-wide CPU statistics.								
ENOSYS	CPU performance counters are inaccessible on this machine. This error can occur when the machine supports CPU performance counters, but some software components are missing. Check to see that all CPU Performance Counter packages have been correctly installed.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr><tr><td>Availability</td><td>SUNWcpcu (32-bit) SUNWcpcux (64-bit)</td></tr><tr><td>Interface Stability</td><td>Evolving</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe	Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	MT-Safe								
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)								
Interface Stability	Evolving								
SEE ALSO	<code>cpc(3CPC)</code> , <code>cpc_seterrfn(3CPC)</code> , <code>proc(4)</code> , <code>attributes(5)</code>								

NAME	cpc_bind_event, cpc_take_sample, cpc_rele – use CPU performance counters on lwps						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lcpc [<i>library...</i>] #include <libcpc.h> int cpc_bind_event(cpc_event_t *event, int flags); int cpc_take_sample(cpc_event_t *event); int cpc_rele(void);</pre>						
DESCRIPTION	Once the events to be sampled have been selected using, for example, <code>cpc_strtoevent(3CPC)</code>, the event selections can be bound to the calling LWP using <code>cpc_bind_event()</code>. If <code>cpc_bind_event()</code> returns successfully, the system has associated performance counter context with the calling LWP. The context allows the system to virtualize the hardware counters to that specific LWP, and the counters are enabled. Two flags are defined that can be passed into the routine to allow the behavior of the interface to be modified, as described below. Counter values can be sampled at any time by calling <code>cpc_take_sample()</code>, and dereferencing the fields of the <code>ce_pic[]</code> array returned. The <code>ce_hrt</code> field contains the timestamp at which the kernel last sampled the counters. To immediately remove the performance counter context on an LWP, the <code>cpc_rele()</code> interface should be used. Otherwise, the context will be destroyed after the LWP or process exits. The caller should take steps to ensure that the counters are sampled often enough to avoid the 32-bit counters wrapping. The events most prone to wrap are those that count processor clock cycles. If such an event is of interest, sampling should occur frequently so that less than 4 billion clock cycles can occur between samples. Practically speaking, this is only likely to be a problem for otherwise idle systems, or when processes are bound to processors, since normal context switching behavior will otherwise hide this problem.						
RETURN VALUES	Upon successful completion, <code>cpc_bind_event()</code> and <code>cpc_take_sample()</code> return 0. Otherwise, these functions return -1, and set <code>errno</code> to indicate the error.						
ERRORS	The <code>cpc_bind_event()</code> and <code>cpc_take_sample()</code> functions will fail if: <table> <tr> <td>EFAULT</td><td>The <i>event</i> argument specifies a bad address.</td></tr> <tr> <td>ENOTSUP</td><td>The caller has attempted an operation that is illegal or not supported on the current platform, such as attempting to specify signal delivery on counter overflow on a CPU that doesn't generate an interrupt on counter overflow.</td></tr> <tr> <td>EAGAIN</td><td>Another process may be sampling system-wide CPU statistics. For <code>cpc_bind_event()</code>, this implies that no new contexts can be created. For <code>cpc_take_sample()</code>, this implies that the performance counter context has been invalidated and must be</td></tr> </table>	EFAULT	The <i>event</i> argument specifies a bad address.	ENOTSUP	The caller has attempted an operation that is illegal or not supported on the current platform, such as attempting to specify signal delivery on counter overflow on a CPU that doesn't generate an interrupt on counter overflow.	EAGAIN	Another process may be sampling system-wide CPU statistics. For <code>cpc_bind_event()</code> , this implies that no new contexts can be created. For <code>cpc_take_sample()</code> , this implies that the performance counter context has been invalidated and must be
EFAULT	The <i>event</i> argument specifies a bad address.						
ENOTSUP	The caller has attempted an operation that is illegal or not supported on the current platform, such as attempting to specify signal delivery on counter overflow on a CPU that doesn't generate an interrupt on counter overflow.						
EAGAIN	Another process may be sampling system-wide CPU statistics. For <code>cpc_bind_event()</code> , this implies that no new contexts can be created. For <code>cpc_take_sample()</code> , this implies that the performance counter context has been invalidated and must be						

cpc_bind_event(3CPC)

	released with <code>cpc_rele()</code>. Robust programs should be coded to expect this behavior and recover from it by releasing the now invalid context by calling <code>cpc_rele()</code> sleeping for a while, then attempting to bind and sample the event once more.
	EINVAL The <code>cpc_take_sample()</code> function has been invoked before the context is bound.
USAGE	Prior to calling <code>cpc_bind_event()</code>, applications should call <code>cpc_access(3CPC)</code> to determine if the counters are accessible on the system.
EXAMPLES	EXAMPLE 1 Use hardware performance counters to measure events in a process. The example below shows how a standalone program can be instrumented with the <code>libcpc</code> routines to use hardware performance counters to measure events in a process. The program performs 20 iterations of a computation, measuring the counter values for each iteration. By default, the example makes the counters measure external cache references and external cache hits; these options are only appropriate for UltraSPARC processors. By setting the <code>PERFEVENTS</code> environment variable to other strings (a list of which can be gleaned from the <code>-h</code> flag of the <code>cpustat</code> or <code>cputrack</code> utilities), other events can be counted. The <code>error()</code> routine below is assumed to be a user-provided routine analogous to the familiar <code>printf(3C)</code> routine from the C library but which also performs an <code>exit(2)</code> after printing the message. <pre>#include <inttypes.h> #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <libcpc.h> int main(int argc, char *argv[]) { int cpuver, iter; char *setting = NULL; cpc_event_t event; if (cpc_version(CPC_VER_CURRENT) != CPC_VER_CURRENT) error("application:library cpc version mismatch!"); if ((cpuver = cpc_getcpuver()) == -1) error("no performance counter hardware!"); if ((setting = getenv("PERFEVENTS")) == NULL) setting = "pic0=EC_ref,pic1=EC_hit"; if (cpc_strtoevent(cpuver, setting, &event) != 0) error("can't measure '%s' on this processor", setting); setting = cpc_eventtostr(&event); if (cpc_access() == -1) error("can't access perf counters: %s", strerror(errno)); if (cpc_bind_event(&event, 0) == -1) error("can't bind lwp%d: %s", _lwp_self(), strerror(errno));</pre>

EXAMPLE 1 Use hardware performance counters to measure events in a process.
(Continued)

```
for (iter = 1; iter <= 20; iter++) {
    cpc_event_t before, after;

    if (cpc_take_sample(&before) == -1)
        break;

    /* ==> Computation to be measured goes here <== */

    if (cpc_take_sample(&after) == -1)
        break;
    (void) printf("%3d: %" PRId64 " %" PRId64 "\n", iter,
        after.ce_pic[0] - before.ce_pic[0],
        after.ce_pic[1] - before.ce_pic[1]);
}

if (iter != 20)
    error("can't sample '%s': %s", setting, strerror(errno));

free(setting);
return (0);
}
```

EXAMPLE 2 Write a signal handler to catch overflow signals.

This example builds on Example 1, but demonstrates how to write the signal handler to catch overflow signals. The counters are preset so that counter zero is 1000 counts short of overflowing, while counter one is set to zero. After 1000 counts on counter zero, the signal handler will be invoked.

First the signal handler:

```
#define PRESET0          (UINT64_MAX - 999ull)
#define PRESET1          0
void
emt_handler(int sig, siginfo_t *sip, void *arg)
{
    ucontext_t *uap = arg;
    cpc_event_t sample;

    if (sig != SIGEMT || sip->si_code != EMT_CPCOVF) {
        psignal(sig, "example");
        psiginfo(sip, "example");
        return;
    }

    (void) printf("lwp%d - si_addr %p ucontext: %%pc %p %%sp %p\n",
        _lwp_self(), (void *)sip->si_addr,
        (void *)uap->uc_mcontext.gregs[PC],
        (void *)uap->uc_mcontext.gregs[USP]);

    if (cpc_take_sample(&sample) == -1)
```

cpc_bind_event(3CPC)

EXAMPLE 2 Write a signal handler to catch overflow signals. (Continued)

```

    error("can't sample: %s", strerror(errno));

(void) printf("0x%" PRIx64 " 0x%" PRIx64 "\n",
    sample.ce_pic[0], sample.ce_pic[1]);
(void) fflush(stdout);

sample.ce_pic[0] = PRESET0;
sample.ce_pic[1] = PRESET1;
if (cpc_bind_event(&sample, CPC_BIND_EMT_OVF) == -1)
    error("cannot bind lwp%d: %s", _lwp_self(), strerror(errno));
}

```

and second the setup code (this can be placed after the code that selects the event to be measured):

```

struct sigaction act;
cpc_event_t event;
...
act.sa_sigaction = emt_handler;
bzero(&act.sa_mask, sizeof (act.sa_mask));
act.sa_flags = SA_RESTART|SA_SIGINFO;
if (sigaction(SIGEMT, &act, NULL) == -1)
    error("sigaction: %s", strerror(errno));
event.ce_pic[0] = PRESET0;
event.ce_pic[1] = PRESET1;
if (cpc_bind_event(&event, CPC_BIND_EMT_OVF) == -1)
    error("cannot bind lwp%d: %s", _lwp_self(), strerror(errno));

for (iter = 1; iter <= 20; iter++) {
    /* ==> Computation to be measured goes here <== */
}

cpc_bind_event(NULL, 0);    /* done */

```

Note that a more general version of the signal handler would use `write(2)` directly instead of depending on the signal-unsafe semantics of `stderr` and `stdout`. Most real signal handlers will probably do more with the samples than just print them out.

NOTES Sometimes, even the overhead of performing a system call will be too disruptive to the events being measured. Once a call to `cpc_bind_event()` has been issued, it is possible to directly access the performance hardware registers from within the application. If the performance counter context is active, then the counters will count on behalf of the current LWP.

SPARC	<code>rd %pic, %rN</code>	! All UltraSPARC
	<code>wr %rN, %pic</code>	! (ditto, but see text)
x86	<code>rdpmc</code>	! Pentium II only

If the counter context is not active or has been invalidated, the `%pic` register (SPARC), and the `rdpmc` instruction (Pentium) will become unavailable.

cpc_bind_event(3CPC)

Note that the two 32-bit UltraSPARC performance counters are kept in the single 64-bit `%pic` register so a couple of additional instructions are required to separate the values. Also note that when the `%pcr` register bit has been set that configures the `%pic` register as readable by an application, it is also writable. Any values written will be preserved by the context switching mechanism.

Pentium II processors support the non-privileged `rdpmc` instruction which requires [5] that the counter of interest be specified in `%ecx`, and returns a 40-bit value in the `%edx:%eax` register pair. There is no non-privileged access mechanism for Pentium I processors.

Handling counter overflow

As described above, when counting events, some processors allow their counter registers to silently overflow. More recent CPUs such as UltraSPARC III and Pentium II, however, are capable of generating an interrupt when the hardware counter overflows. Some processors offer more control over when interrupts will actually be generated. For example, they might allow the interrupt to be programmed to occur when only one of the counters overflows. See `cpc_strtoevent(3CPC)` for the syntax.

The most obvious use for this facility is to ensure that the full 64-bit counter values are maintained without repeated sampling. However, current hardware does not record which counter overflowed. A more subtle use for this facility is to preset the counter to a value to a little less than the maximum value, then use the resulting interrupt to catch the counter overflow associated with that event. The overflow can then be used as an indication of the frequency of the occurrence of that event.

Note that the interrupt generated by the processor may not be particularly precise. That is, the particular instruction that caused the counter overflow may be earlier in the instruction stream than is indicated by the program counter value in the `ucontext`.

When `cpc_bind_event()` is called with the `CPC_BIND_EMT_OVF` flag set, then as before, the control registers and counters are preset from the 64-bit values contained in `event`. However, when the flag is set, the kernel arranges to send the calling process a `SIGEMT` signal when the overflow occurs, with the `si_code` field of the corresponding `siginfo` structure set to `EMT_CPCOVF`, and the `si_addr` field is the program counter value at the time the overflow interrupt was delivered. Counting, and thus the subsequent delivery of the signal on overflow is disabled until the next call to `cpc_bind_event()`. Even in a multithreaded process, during execution of the signal handler, the thread behaves as if it is temporarily bound to the running LWP.

Different processors have different counter ranges available, though all processors supported by Solaris allow at least 31 bits to be specified as a counter preset value; thus portable preset values lie in the range `UINT64_MAX` to `UINT64_MAX-INT32_MAX`.

cpc_bind_event(3CPC)

The appropriate preset value will often need to be determined experimentally. Typically, it will depend on the event being measured, as well as the desire to minimize the impact of the act of measurement on the event being measured; less frequent interrupts and samples lead to less perturbation of the system.

If the processor cannot detect counter overflow, this call will fail (ENOTSUP). Specifying a null event unbinds the context from the underlying LWP and disables signal delivery. Currently, only user events can be measured using this technique. See Example 2, above.

**Inheriting events
onto multiple
LWPs**

By default, the library binds the performance counter context to the current LWP only. If the CPC_BIND_LWP_INHERIT flag is set, then any subsequent LWPs created by that LWP will automatically inherit the same performance counter context. The counters will be initialized to 0 as if a cpc_bind_event() had just been issued. This automatic inheritance behavior can be useful when dealing with multithreaded programs to determine aggregate statistics for the program as a whole.

If the CPC_BIND_EMT_OVF flag is also set, the process will immediately dispatch a SIGEMT signal to the freshly created LWP so that it can preset its counters appropriately on the new LWP. This initialization condition can be detected using cpc_take_sample() to check that both ce_pic[] values are set to UINT64_MAX.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO

cpustat(1), cpc(3CPC), cpc_access(3CPC), cpc_strtoevent(3CPC), attributes(5)

NAME	cpc_count_usr_events, cpc_count_sys_events – enable and disable performance counters				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lcpc [<i>library...</i>] #include <libcpc.h> int cpc_count_usr_events(int <i>enable</i>) ; int cpc_count_sys_events(int <i>enable</i>) ;</pre>				
DESCRIPTION	In certain applications, it can be useful to explicitly enable and disable performance counters at different times so that the performance of a critical algorithm can be examined. The <code>cpc_count_usr_events()</code> function can be used to control whether events are counted on behalf of the application running in user mode, while <code>cpc_count_sys_events()</code> can be used to control whether events are counted on behalf of the application while it is running in the kernel, without otherwise disturbing the binding of events to the invoking LWP. If the <i>enable</i> argument is non-zero, counting of events is enabled, otherwise they are disabled.				
RETURN VALUES	Upon successful completion, <code>cpc_count_usr_events()</code> and <code>cpc_count_sys_events()</code> return 0. Otherwise, the functions return -1 and set <code>errno</code> to indicate the error.				
ERRORS	The <code>cpc_count_usr_events()</code> and <code>cpc_count_sys_events()</code> functions will fail if: <table> <tr> <td>EAGAIN</td><td>The associated performance counter context has been invalidated by another process.</td></tr> <tr> <td>EINVAL</td><td>No performance counter context has been created, or an attempt was made to enable system events while delivering counter overflow signals.</td></tr> </table>	EAGAIN	The associated performance counter context has been invalidated by another process.	EINVAL	No performance counter context has been created, or an attempt was made to enable system events while delivering counter overflow signals.
EAGAIN	The associated performance counter context has been invalidated by another process.				
EINVAL	No performance counter context has been created, or an attempt was made to enable system events while delivering counter overflow signals.				
EXAMPLES	EXAMPLE 1 Use <code>cpc_count_usr_events()</code> to minimize code needed by application. In this example, the routine <code>cpc_count_usr_events()</code> is used to minimize the amount of code that needs to be added to the application. The <code>cputrack(1)</code> command can be used in conjunction with these interfaces to provide event programming, sampling, and reporting facilities. If the application is instrumented in this way and then started by <code>cputrack</code> with the <code>nouser</code> flag set in the event specification, counting of user events will only be enabled around the critical code section of interest. If the program is run normally, no harm will ensue. <pre>int have_counters = 0; int main(int argc, char *argv[]) { if (cpc_version(CPC_VER_CURRENT) == CPC_VER_CURRENT && cpc_getcpuver() != -1 && cpc_access() == 0) have_counters = 1;</pre>				

cpc_count_usr_events(3CPC)

EXAMPLE 1 Use `cpc_count_usr_events()` to minimize code needed by application.
(Continued)

```
/* ... other application code */

if (have_counters)
    (void) cpc_count_usr_events(1);

/* ==> Code to be measured goes here <== */

if (have_counters)
    (void) cpc_count_usr_events(0);

/* ... other application code */
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO `cputrack(1)`, `cpc(3CPC)`, `cpc_access(3CPC)`, `cpc_version(3CPC)`,
`cpc_getcpuver(3CPC)`, `cpc_bind_event(3CPC)`, `cpc_pctx_bind_event(3CPC)`,
`attributes(5)`

NAME	cpc_event – data structure to describe CPU performance counters
SYNOPSIS	<pre>#include <libcpc.h></pre>
DESCRIPTION	The libcpc interfaces manipulate CPU performance counters using the cpc_event_t data structure. This structure contains several fields that are common to all processors, and some that are processor-dependent. These structures can be declared by a consumer of the API, thus the size and offsets of the fields and the entire data structure are fixed per processor for any particular version of the library. See cpc_version(3CPC) for details of library versioning.
SPARC	For UltraSPARC, the structure contains the following members: <pre>typedef struct { int ce_cpuver; hrtime_t ce_hrt; uint64_t ce_tick; uint64_t ce_pic[2]; uint64_t ce_pcr; } cpc_event_t;</pre>
x86	For Pentium, the structure contains the following members: <pre>typedef struct { int ce_cpuver; hrtime_t ce_hrt; uint64_t ce_tsc; uint64_t ce_pic[2]; uint32_t ce_pes[2]; #define ce_cesr ce_pes[0] } cpc_event_t;</pre> The APIs are used to manipulate the highly processor-dependent control registers (the ce_pcr, ce_cesr, and ce_pes fields); the programmer is strongly advised not to reference those fields directly in portable code. The ce_pic array elements contain 64-bit accumulated counter values. The hardware registers are virtualized to 64-bit quantities even though the underlying hardware only supports 32-bits (UltraSPARC) or 40-bits (Pentium) before overflow. The ce_hrt field is a high resolution timestamp taken at the time the counters were sampled by the kernel. This uses the same timebase as gethrtime(3C). On SPARC V9 machines, the number of cycles spent running on the processor is computed from samples of the processor-dependent %tick register, and placed in the ce_tick field. On Pentium processors, the processor-dependent time-stamp counter register is similarly sampled and placed in the ce_tsc field.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE

cpc_event(3CPC)

Availability	SUNWcpcu

SEE ALSO gethrtime(3C), cpc(3CPC), cpc_version(3CPC), attributes(5).

NAME	cpc_event_diff, cpc_event_accum – simple difference and accumulate operations						
SYNOPSIS	<pre>cc [flag...] file... -lcpc [library...] #include <libcpc.h> void cpc_event_accum(cpc_event_t *accum, cpc_event_t *event); void cpc_event_diff(cpc_event_t *diff, cpc_event_t *after, cpc_event_t *before);</pre>						
DESCRIPTION	The <code>cpc_event_accum()</code> and <code>cpc_event_diff()</code> functions perform common accumulate and difference operations on <code>cpc_event(3CPC)</code> data structures. Use of these functions increases program portability, since structure members are not referenced directly.						
cpc_event_accum()	The <code>cpc_event_accum()</code> function adds the <code>ce_pic</code> fields of <i>event</i> into the corresponding fields of <i>accum</i>. The <code>ce_hrt</code> field of <i>accum</i> is set to the later of the times in <i>event</i> and <i>accum</i>. SPARC: The function adds the contents of the <code>ce_tick</code> field of <i>event</i> into the corresponding field of <i>accum</i>. x86: The function adds the contents of the <code>ce_tsc</code> field of <i>event</i> into the corresponding field of <i>accum</i>.						
cpc_event_diff()	The <code>cpc_event_diff()</code> function places the difference between the <code>ce_pic</code> fields of <i>after</i> and <i>before</i> and places them in the corresponding field of <i>diff</i>. The <code>ce_hrt</code> field of <i>diff</i> is set to the <code>ce_hrt</code> field of <i>after</i>. SPARC: Additionally, the function computes the difference between the <code>ce_tick</code> fields of <i>after</i> and <i>before</i>, and places it in the corresponding field of <i>diff</i>. x86: Additionally, the function computes the difference between the <code>ce_tsc</code> fields of <i>after</i> and <i>before</i>, and places it in the corresponding field of <i>diff</i>.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> <tr> <td>Availability</td><td>SUNWcpcu (32-bit) SUNWcpcux (64-bit)</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe	Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	MT-Safe						
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)						

cpc_event_diff(3CPC)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `cpc(3CPC)`, `cpc_event(3CPC)`, `attributes(5)`.

NAME	cpc_getcpuver, cpc_getcciname, cpc_getcpuref, cpc_getusage, cpc_getnpic, cpc_walk_names – determine CPU performance counter configuration
SYNOPSIS	<pre>cc [flag...] file... -lcpc [library...] #include <libcpc.h> int cpc_getcpuver(void); const char *cpc_getcciname(int cpuver); const char *cpc_getcpuref(int cpuver); const char *cpc_getusage(int cpuver); uint_t cpc_getnpic(int cpuver); void cpc_walk_names(int cpuver, int regno, void *arg, void (*action)(void *arg, int regno, const char *name, uint8_t bits));</pre>
DESCRIPTION	The <code>cpc_getcpuver()</code> function returns an abstract integer that corresponds to the distinguished version of the underlying processor. The library distinguishes between processors solely on the basis of their support for performance counters, so the version returned should not be interpreted in any other way. The set of values returned by the library is unique across all processor implementations. The <code>cpc_getcpuver()</code> function returns <code>-1</code> if the library cannot support CPU performance counters on the current architecture. This may be because the processor has no such counter hardware, or because the library is unable to recognize it. Either way, such a return value indicates that the configuration functions described on this manual page cannot be used. The <code>cpc_getcciname()</code> function returns a printable description of the processor performance counter interfaces—for example, the string <i>UltraSPARC I&II</i>. Note that this name should not be assumed to be the same as the name the manufacturer might otherwise ascribe to the processor. It simply names the performance counter interfaces as understood by the library, and thus names the set of performance counter events that can be described by that interface. If the <code>cpuver</code> argument is unrecognized, the function returns <code>NULL</code>. The <code>cpc_getcpuref()</code> function returns a string that describes a reference work that should be consulted to (allow a human to) understand the semantics of the performance counter events that are known to the library. If the <code>cpuver</code> argument is unrecognized, the function returns <code>NULL</code>. The string returned might be substantially longer than 80 characters. Callers printing to a terminal might want to insert line breaks as appropriate. The <code>cpc_getusage()</code> function returns a compact description of the <code>getsubopt()</code>-oriented syntax that is consumed by <code>cpc_strtoevent(3CPC)</code>. It is returned as a space-separated set of tokens to allow the caller to wrap lines at convenient boundaries. If the <code>cpuver</code> argument is unrecognized, the function returns <code>NULL</code>.

cpc_getcpuver(3CPC)

The `cpc_getnpic()` function returns the number of valid fields in the `ce_pic[]` array of a `cpc_event_t` data structure.

The library maintains a list of events that it believes the processor capable of measuring, along with the bit patterns that must be set in the corresponding control register, and which counter the result will appear in. The `cpc_walk_names()` function calls the `action()` function on each element of the list so that an application can print appropriate help on the set of events known to the library. The `arg` parameter is passed uninterpreted from the caller on each invocation of the `action()` function.

If the parameters specify an invalid or unknown CPU or register number, the function silently returns without invoking the action function.

USAGE Prior to calling any of these functions, applications should call `cpc_access(3CPC)` to determine if the counters are accessible on the system.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO `cpc(3CPC)`, `cpc_access(3CPC)`, `attributes(5)`.

NOTES Only SPARC processors are described by the SPARC version of the library, and only x86 processors are described by the x86 version of the library.

NAME	cpc_pctx_bind_event, cpc_pctx_take_sample, cpc_pctx_rele, cpc_pctx_invalidate – access CPU performance counters in other processes						
SYNOPSIS	<pre>cc [flag...] file... -lcpc -lpctx [library...] #include <libpctx.h> #include <libcpc.h> int cpc_pctx_bind_event(pctx_t *pctx, id_t lwpid, cpc_event_t *event, int flags); int cpc_pctx_take_sample(pctx_t *pctx, id_t lwpid, cpc_event_t *event); int cpc_pctx_rele(pctx_t *pctx, id_t lwpid); int cpc_pctx_invalidate(pctx_t *pctx, id_t lwpid);</pre>						
DESCRIPTION	These functions are designed to be run in the context of an event handler created using the libpctx(3LIB) family of functions that allow the caller, also known as the <i>controlling process</i>, to manipulate the performance counters in the context of a <i>controlled process</i>. The controlled process is described by the <i>pctx</i> argument, which must be obtained from an invocation of <code>pctx_capture(3CPC)</code> or <code>pctx_create(3CPC)</code> and passed to the functions described on this page in the context of an event handler. The semantics of the functions <code>cpc_pctx_bind_event()</code>, <code>cpc_pctx_take_sample()</code>, and <code>cpc_pctx_rele()</code> are directly analogous to those of <code>cpc_bind_event()</code>, <code>cpc_take_sample()</code>, and <code>cpc_rele()</code> described on the <code>cpc_bind_event(3CPC)</code> manual page. The <code>cpc_pctx_invalidate()</code> function allows the performance context to be invalidated in an LWP in the controlled process.						
RETURN VALUES	These functions return 0 on success. On failure, they return -1 and set <code>errno</code> to indicate the error.						
ERRORS	The <code>cpc_pctx_bind_event()</code>, <code>cpc_pctx_take_sample()</code>, and <code>cpc_pctx_rele()</code> functions return the same <code>errno</code> values the analogous functions described on the <code>cpc_bind_event(3CPC)</code> manual page. In addition, these function may fail if: <table> <tr> <td>ESRCH</td><td>The value of the <i>lwpid</i> argument is invalid in the context of the controlled process.</td></tr> </table>	ESRCH	The value of the <i>lwpid</i> argument is invalid in the context of the controlled process.				
ESRCH	The value of the <i>lwpid</i> argument is invalid in the context of the controlled process.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> <tr> <td>Availability</td><td>SUNWcpcu (32-bit)</td></tr> </table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe	Availability	SUNWcpcu (32-bit)
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Unsafe						
Availability	SUNWcpcu (32-bit)						

cpc_pctx_bind_event(3CPC)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
	SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO cpc(3CPC), cpc_bind_event(3CPC), pctx_capture(3CPC), pctx_create(3CPC), attributes(5).

NOTES The capability to create and analyze overflow events in other processes is not available, though it may be made available in a future version of this API. In the current implementation, the *flags* field must be specified as 0.

NAME	cpc_seterrfn – control libcpc error reporting								
SYNOPSIS	<pre>cc [<i>flag</i>...] <i>file</i>... -lcpc [<i>library</i>...]#include <libcpc.h> typedef void(cpc_errfn_t) (const char *<i>fn</i>, const char *<i>fmt</i>, va_list <i>ap</i>) ; void cpc_seterrfn(cpc_errfn_t *<i>errfn</i>) ;</pre>								
DESCRIPTION	For the convenience of programmers instrumenting their code, several libcpc functions automatically emit to <code>stderr</code> error messages that attempt to provide a more detailed explanation of their error return values. While this can be useful for simple programs, some applications may wish to report their errors differently—for example, to a window or to a log file. The <code>cpc_seterrfn()</code> function allows the caller to provide an alternate function for reporting errors; the type signature is shown above. The <i>fn</i> argument is passed the library function name that detected the error, the format string <i>fmt</i> and argument pointer <i>ap</i> can be passed directly to <code>vsnprintf(3C)</code> or similar <code>varargs</code>-based routine for formatting. The default printing routine can be restored by calling the routine with an <i>errfn</i> argument of <code>NULL</code>.								
EXAMPLES	EXAMPLE 1 Debugging example. This example produces error messages only when debugging the program containing it, or when the <code>cpc_strtoevent()</code> function is reporting an error when parsing an event specification <pre>int debugging; void myapp_errfn(const char *fn, const char *fmt, va_list ap) { if (strcmp(fn, "strtoevent") != 0 && !debugging) return; (void) fprintf(stderr, "myapp: cpc_%s(): ", fn); (void) vfprintf(stderr, fmt, ap); }</pre>								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> <tr> <td>Availability</td><td>SUNWcpcu (32-bit) SUNWcpcux (64-bit)</td></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe	Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Unsafe								
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)								
Interface Stability	Evolving								
SEE ALSO	<code>cpc(3CPC)</code> , <code>vsnprintf(3C)</code> , <code>attributes(5)</code> .								

cpc_shared_open(3CPC)

NAME	cpc_shared_open, cpc_shared_bind_event, cpc_shared_take_sample, cpc_shared_rele, cpc_shared_close – use CPU performance counters on processors
SYNOPSIS	<pre>cc [flag...] file... -lcpc [library...] #include <libcpc.h> int cpc_shared_open(void); int cpc_shared_bind_event(int fd, cpc_event_t *event, int flags); int cpc_shared_take_sample(int fd, cpc_event_t *event); int cpc_shared_rele(int fd); void cpc_shared_close(int fd);</pre>
DESCRIPTION	The <code>cpc_shared_open()</code> function allows the caller to access the hardware counters in such a way that the performance of the currently bound CPU can be measured. The function returns a file descriptor if successful. Only one such open can be active at a time on any CPU. The <code>cpc_shared_bind_event()</code>, <code>cpc_shared_take_sample()</code>, and <code>cpc_shared_rele()</code> functions are directly analogous to the corresponding <code>cpc_bind_event()</code>, <code>cpc_take_sample()</code>, and <code>cpc_rele()</code> functions described on the <code>cpc_bind_event(3CPC)</code> manual page, except that they operate on the counters of a particular processor.
USAGE	If a thread wishes to access the counters using this interface, it must do so using a thread bound to an lwp, (see the <code>THR_BOUND</code> flag to <code>thr_create(3THR)</code>), that has in turn bound itself to a processor using <code>processor_bind(2)</code>. Unlike the <code>cpc_bind_event(3CPC)</code> family of functions, no counter context is attached to those lwps, so the performance counter samples from the processors reflects the system-wide usage, instead of per-lwp usage. The first successful invocation of <code>cpc_shared_open()</code> will immediately invalidate <i>all</i> existing performance counter context on the system, and prevent <i>all</i> subsequent attempts to bind counter context to lwps from succeeding anywhere on the system until the last caller invokes <code>cpc_shared_close()</code>. This is because it is impossible to simultaneously use the counters to accurately measure per-lwp and system-wide events, so there is an exclusive interlock between these uses. Access to the shared counters is mediated by file permissions on a <code>cpc</code> pseudo device. As shipped, only the superuser is allowed to access the shared device; this is because doing so prevents use of the counters on a per-lwp basis to any other users. The <code>CPC_BIND_LWP_INHERIT</code> and <code>CPC_BIND_EMT_OVF</code> flags are invalid for the shared interface.
RETURN VALUES	On success, the functions (apart from <code>cpc_shared_close()</code>) return 0. On failure, the functions return -1 and set <code>errno</code> , to indicate the reason.

cpc_shared_open(3CPC)

ERRORS	ENXIO	The current machine either has no performance counters, or has been configured to disallow access to them system-wide.
	EACCES	The caller does not have appropriate privilege to access the CPU performance counters system-wide.
	EAGAIN	For cpc_shared_open(), this value implies that the counters on the bound cpu are busy because they are already being used to measure system-wide events by some other caller.
	EAGAIN	Otherwise, this return value implies that the counters are not available because the thread has been unbound from the processor it was bound to at open time. Robust programs should be coded to expect this behavior, and should invoke cpc_shared_close(), before retrying the operation.
	EINVAL	The counters cannot be accessed on the current CPU because the calling thread is not bound to that CPU using processor_bind(2).
	EFAULT	The <i>event</i> argument specifies a bad address.
	ENOTSUP	The caller has attempted an operation that is illegal or not supported on the current platform.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWcpcu (32-bit)
	SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO processor_bind(2), cpc(3CPC), cpc_bind_event(3CPC), thr_create(3THR), attributes(5)

cpc_strtoevent(3CPC)

NAME	cpc_strtoevent, cpc_eventtostr – translate strings to and from events
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lcpc [<i>library...</i>] #include <libcpc.h> int cpc_strtoevent(int <i>cpuver</i>, const char *<i>spec</i>, cpc_event_t *<i>event</i>); char *cpc_eventtostr(cpc_event_t *<i>event</i>);</pre>
DESCRIPTION	The <code>cpc_strtoevent()</code> function translates an event specification to the appropriate collection of control bits in a <code>cpc_event_t</code> structure pointed to by the <i>event</i> argument. The event specification is a getsubopt(3C)–style string that describes the event and any attributes that the processor can apply to the event or events. If successful, the function returns 0, the <code>ce_cpuver</code> field and the ISA-dependent control registers of event are initialized appropriately, and the rest of the <code>cpc_event_t</code> structure is initialized to 0. The <code>cpc_eventtostr()</code> function takes an event and constructs a compact canonical string representation for that event.
RETURN VALUES	Upon successful completion, <code>cpc_strtoevent()</code> returns 0. If the string cannot be decoded, a non-zero value is returned and a message is printed using the library's error-reporting mechanism (see <code>cpc_seterrfn(3CPC)</code>). Upon successful completion, <code>cpc_eventtostr()</code> returns a pointer to a string. The string returned must be freed by the caller using <code>free(3C)</code>. If <code>cpc_eventtostr()</code> a null pointer is returned.
USAGE	The event selection syntax used is processor architecture-dependent. The supported processor families allow variations on how events are counted as well as what events can be counted. This information is available in compact form from the <code>cpc_getusage()</code> function (see <code>cpc_getcpuver(3CPC)</code>), but is explained in further detail below.
UltraSPARC	On UltraSPARC processors, the syntax for setting options is as follows: <pre>pic0=<eventspec>,pic1=<eventspec> [,sys] [,nouser]</pre> This syntax, which reflects the simplicity of the options available using the <code>%pcr</code> register, forces both counter events to be selected. By default only user events are counted; however, the <code>sys</code> keyword allows system (kernel) events to be counted as well. User event counting can be disabled by specifying the <code>nouser</code> keyword. The keywords <code>pic0</code> and <code>pic1</code> may be omitted; they can be used to resolve ambiguities if they exist.
Pentium I	On Pentium processors, the syntax for setting counter options is as follows: <pre>pic0=<eventspec>,pic1=<eventspec> [,sys[[0 1]]] [,nouser[[0 1]]] [,noedge[[0 1]]] [,pc[[0 1]]]</pre>

The syntax and semantics are the same as UltraSPARC, except that it is possible to specify whether a particular counter counts user or system events. If unspecified, the specification is presumed to apply to both counters.

There are some additional keywords. The `noedge` keyword specifies that the counter should count clocks (duration) instead of events. The `pc` keyword allows the external pin control pins to be set high (defaults to low). When the pin control register is set high, the external pin will be asserted when the associated register overflows. When the pin control register is set low, the external pin will be asserted when the counter has been incremented. The electrical effect of driving the pin is dependent upon how the motherboard manufacturer has chosen to connect it, if it is connected at all.

Pentium II For Pentium II processors, the syntax is substantially more complex, reflecting the complex configuration options available:

```
pic0=<eventspec>,pic1=<eventspec> [,sys[[0|1]]]
[,nouser[[0|1]]] [,noedge[[0|1]]] [,pc[[0|1]]] [,inv[[0|1]]] [,int[[0|1]]]
[,cmask[0|1]=<maskspec>] [,umask[0|1]=<maskspec>]
```

This syntax is a straightforward extension of the earlier syntax. The additional `inv`, `int`, `cmask0`, `cmask1`, `umask0`, and `umask1` keywords allow extended counting semantics. The mask specification is a number between 0 and 255, expressed in hexadecimal, octal or decimal notation.

EXAMPLES

SPARC **EXAMPLE 1** SPARC Example.

```
cpc_event_t event;
char *setting = "pic0=EC_ref,pic1=EC_hit"; /* UltraSPARC-specific */

if (cpc_strtoevent(cpuver, setting, &event) != 0)
    /* can't measure 'setting' on this processor */
else
    setting = cpc_eventtostr(&event);
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO `cpc(3CPC)`, `cpc_getcpuver(3CPC)`, `cpc_seterrfn(3CPC)`, `free(3C)`, `getsubopt(3C)`, `attributes(5)`

cpc_strtoevent(3CPC)

NOTES	These functions are provided as a convenience only. As new processors are usually released asynchronously with software, the library allows the <code>pic0</code> and <code>pic1</code> keywords to interpret numeric values specified directly in hexadecimal, octal, or decimal.

NAME	cpc_version – coordinate CPC library and application versions								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lcpc [<i>library...</i>] #include <libcpc.h> uint_t cpc_version(uint_t <i>version</i>);</pre>								
DESCRIPTION	The <code>cpc_version()</code> function takes an interface version as an argument and returns an interface version as a result. Usually, the argument will be the value of <code>CPC_VER_CURRENT</code> bound to the application when it was compiled.								
RETURN VALUES	If the version requested is still supported by the implementation, <code>cpc_version()</code> returns the requested version number and the application can use the facilities of the library on that platform. If the implementation cannot support the version needed by the application, <code>cpc_version()</code> returns <code>CPC_VER_NONE</code>, indicating that the application will at least need to be recompiled to operate correctly on the new platform, and may require further changes. If <i>version</i> is <code>CPC_VER_NONE</code>, <code>cpc_version()</code> returns the most current version of the library.								
EXAMPLES	EXAMPLE 1 Protect an application from using an incompatible library. The following lines of code protect an application from using an incompatible library: <pre>if (cpc_version(CPC_VER_CURRENT) == CPC_VER_NONE) { /* version mismatch - library cannot translate */ exit(1); }</pre>								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> <tr> <td>Availability</td><td>SUNWcpcu (32-bit) SUNWcpcux (64-bit)</td></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe	Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Unsafe								
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)								
Interface Stability	Evolving								
SEE ALSO	<code>cpc(3CPC)</code> , <code>attributes(5)</code>								
NOTES	The version number is used only to express incompatible semantic changes in the performance counter interfaces on the given platform within a single instruction set architecture, for example, when a new set of performance counter registers are added to an existing processor family that cannot be specified in the existing <code>cpc_event_t</code> data structure.								

demangle(3EXT)

NAME	demangle, cplus_demangle – decode a C++ encoded symbol name						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> [<i>library</i> ...] -ldemangle #include <demangle.h> int cplus_demangle(const char *<i>symbol</i>, char *<i>prototype</i>, size_t <i>size</i>);</pre>						
DESCRIPTION	The <code>cplus_demangle()</code> function decodes (demangles) a C++ linker symbol name (mangled name) into a (partial) C++ prototype, if possible. C++ mangled names may not have enough information to form a complete prototype. The <i>symbol</i> string argument points to the input mangled name. The <i>prototype</i> argument points to a user-specified output string buffer, of size bytes. The <code>cplus_demangle()</code> function operates on mangled names generated by SPARCompilers C++ 3.0.1, 4.0.1, 4.1 and 4.2. The <code>cplus_demangle()</code> function improves and replaces the <code>demangle()</code> function. Refer to the <code>CC.1</code>, <code>dem.1</code>, and <code>c++filt.1</code> manual pages in the <code>/opt/SUNWspro/man/man1</code> directory. These pages are only available with the SPROcc package.						
RETURN VALUES	The <code>cplus_demangle()</code> function returns the following values: <table><tr><td>0</td><td>The <i>symbol</i> argument is a valid mangled name and <i>prototype</i> contains a (partial) prototype for the symbol.</td></tr><tr><td>DEMANGLE_ENAME</td><td>The <i>symbol</i> argument is not a valid mangled name and the content of <i>prototype</i> is a copy of the symbol.</td></tr><tr><td>DEMANGLE_ESPACE</td><td>The <i>prototype</i> output buffer is too small to contain the prototype (or the symbol), and the content of <i>prototype</i> is undefined.</td></tr></table>	0	The <i>symbol</i> argument is a valid mangled name and <i>prototype</i> contains a (partial) prototype for the symbol.	DEMANGLE_ENAME	The <i>symbol</i> argument is not a valid mangled name and the content of <i>prototype</i> is a copy of the symbol.	DEMANGLE_ESPACE	The <i>prototype</i> output buffer is too small to contain the prototype (or the symbol), and the content of <i>prototype</i> is undefined.
0	The <i>symbol</i> argument is a valid mangled name and <i>prototype</i> contains a (partial) prototype for the symbol.						
DEMANGLE_ENAME	The <i>symbol</i> argument is not a valid mangled name and the content of <i>prototype</i> is a copy of the symbol.						
DEMANGLE_ESPACE	The <i>prototype</i> output buffer is too small to contain the prototype (or the symbol), and the content of <i>prototype</i> is undefined.						

devid_get(3DEVID)

NAME	devid_get, devid_compare, devid_deviceid_to_nmlist, devid_free, devid_free_nmlist, devid_get_minor_name, devid_sizeof, devid_str_decode, devid_str_free, devid_str_encode, devid_valid – device ID interfaces for user applications
SYNOPSIS	<pre>cc [flag...] file... -ldevid [library...] #include <devid.h> int devid_get(int fd, ddi_devid_t *retdevid); void devid_free(ddi_devid_t devid); int devid_get_minor_name(int fd, char **retminor_name); int devid_deviceid_to_nmlist(char *search_path, ddi_devid_t devid, char *minor_name, devid_nmlist_t **retlist); void devid_free_nmlist(devid_nmlist_t *list); int devid_compare(ddi_devid_t devid1, ddi_devid_t devid2); size_t devid_sizeof(ddi_devid_t devid); int devid_valid(ddi_devid_t devid); char *devid_str_encode(ddi_devid_t devid, char *minor_name); int devid_str_decode(char *devidstr, ddi_devid_t *retdevid, char **retminor_name); void devid_str_free(char *str);</pre>
DESCRIPTION	These functions provide unique identifiers (device IDs) for devices. Applications and device drivers use these functions to identify and locate devices, independent of the device's physical connection or its logical device name or number. The <code>devid_get()</code> function returns in <code>retdevid</code> the device ID for the device associated with the open file descriptor <code>fd</code>, which refers to any device. It returns an error if the device does not have an associated device ID. The caller must free the memory allocated for <code>retdevid</code> using the <code>devid_free()</code> function. The <code>devid_free()</code> function frees the space that was allocated for the returned <code>devid</code> by <code>devid_get()</code> and <code>devid_str_decode()</code>. The <code>devid_get_minor_name()</code> function returns the minor name, in <code>retminor_name</code>, for the device associated with the open file descriptor <code>fd</code>. This name is specific to the particular minor number, but is "instance number" specific. The caller of this function must free the memory allocated for the returned <code>retminor_name</code> string using <code>devid_str_free()</code>. The <code>devid_deviceid_to_nmlist()</code> function returns an array of <code>devid_nmlist</code> structures, where each entry matches the <code>devid</code> and <code>minor_name</code> passed in. If the <code>minor_name</code> specified is one of the special values (<code>DEVID_MINOR_NAME_ALL</code>, <code>DEVID_MINOR_NAME_ALL_CHR</code>, or <code>DEVID_MINOR_NAME_ALL_BLK</code>), then all minor names associated with <code>devid</code> which also meet the special <code>minor_name</code> filtering

devid_get(3DEVID)

requirements are returned. The *devid_nmlist* structure contains the device name and device number. The last entry of the array contains a null pointer for the *devname* and *NODEV* for the device number. This function traverses the file tree, starting at *search_path*. For each device with a matching device ID and minor name tuple, a device name and device number are added to the *retlist*. If no matches are found, an error is returned. The caller of this function must free the memory allocated for the returned array with the *devid_free_nmlist()* function. This function may take a long time to complete if called with the device ID of an unattached device.

The *devid_free_nmlist()* function frees the memory allocated by the *devid_deviceid_to_nmlist()* function.

The *devid_compare()* function compares two device IDs and determines both equality and sort order. The function returns an integer greater than 0 if the device ID pointed to by *devid1* is greater than the device ID pointed to by *devid2*. It returns 0 if the device ID pointed to by *devid1* is equal to the device ID pointed to by *devid2*. It returns an integer less than 0 if the device ID pointed to by *devid1* is less than the device ID pointed to by *devid2*. This function is the only valid mechanism to determine the equality of two devids. This function may indicate equality for arguments which by simple inspection appear different.

The *devid_sizeof()* function returns the size of *devid* in bytes.

The *devid_valid()* function validates the format of a *devid*. It returns 1 if the format is valid, and 0 if invalid. This check may not be as complete as the corresponding kernel function *ddi_devid_valid()* (see *ddi_devid_compare(9F)*).

The *devid_str_encode()* function encodes a *devid* and *minor_name* into a null-terminated ASCII string, returning a pointer to that string. To avoid shell conflicts, the *devid* portion of the string is limited to uppercase and lowercase letters, digits, and the plus (+), minus (-), period (.), equals (=), underscore (_), tilde (~), and comma (,) characters. If there is an ASCII quote character in the binary form of a *devid*, the string representation will be in *hex_id* form, not *ascii_id* form. The comma (,) character is added for "id1," at the head of the string *devid*. If both a *devid* and a *minor_name* are non-null, a slash (/) is used to separate the *devid* from the *minor_name* in the encoded string. If *minor_name* is null, only the *devid* is encoded. If the *devid* is null then the special string "id0" is returned. Note that you cannot compare the returned string against another string with *strcmp(3C)* to determine devid equality. The string returned must be freed by calling *devid_str_free()*.

The *devid_str_decode()* function takes a string previously produced by the *devid_str_encode()* or *ddi_devid_str_encode()* (see *ddi_devid_compare(9F)*) function and decodes the contained device ID and minor name, allocating and returning pointers to the extracted parts via the *retdevid* and *retminor_name* arguments. If the special *devidstr* "id0" was specified, the returned device ID and minor name will both be null. A non-null returned devid must be freed by the caller by the *devid_free()* function. A non-null returned minor name must be freed by calling *devid_str_free()*.

devid_get(3DEVID)

RETURN VALUES

The `devid_str_free()` function frees the character string returned by `devid_str_encode()` and the `retminor_name` argument returned by `devid_str_decode()`.

Upon successful completion, the `devid_get()`, `devid_get_minor_name()`, `devid_str_decode()`, and `devid_deviceid_to_nmlist()` functions return 0. Otherwise, they return -1.

The `devid_compare()` function returns the following values:

- 1 The device ID pointed to by *devid1* is less than the device ID pointed to by *devid2*.
- 0 The device ID pointed to by *devid1* is equal to the device ID pointed to by *devid2*.
- 1 The device ID pointed to by *devid1* is greater than the device ID pointed to by *devid2*.

The `devid_sizeof()` function returns the size of *devid* in bytes. If *devid* is null, the number of bytes that must be allocated and initialized to determine the size of a complete device ID is returned.

The `devid_valid()` function returns 1 if the *devid* is valid and 0 if the *devid* is invalid.

The `devid_str_encode()` function returns NULL to indicate failure. Failure may be caused by attempting to encode an invalid string. If the return value is non-null, the caller must free the returned string by using the `devid_str_free()` function.

EXAMPLES

EXAMPLE 1 Using `devid_get()`, `devid_get_minor_name()`, and `devid_str_encode()`

The following example shows the proper use of `devid_get()`, `devid_get_minor_name()`, and `devid_str_encode()` to free the space allocated for *devid*, *minor_name* and encoded *devid*.

```
int fd;
ddi_devid_t   devid;
char          *minor_name, *devidstr;
if ((fd = open("/dev/dsk/c0t3d0s0", O_RDONLY|O_NDELAY)) < 0) {
    ...
}
if (devid_get(fd, &devid) != 0) {
    ...
}
if (devid_get_minor_name(fd, &minor_name) != 0) {
    ...
}
if ((devidstr = devid_str_encode(devid, minor_name)) == 0) {
    ...
}
printf("devid %s\
```

devid_get(3DEVID)

EXAMPLE 1 Using `devid_get()`, `devid_get_minor_name()`, and `devid_str_encode()` *(Continued)*

```
", devidstr);
devid_str_free(devidstr);
devid_free(devid);
devid_str_free(minor_name);
```

EXAMPLE 2 Using `devid_deviceid_to_nmlist()` and `devid_free_nmlist()`

The following example shows the proper use of `devid_deviceid_to_nmlist()` and `devid_free_nmlist()`:

```
devid_nmlist_t *list = NULL;
int err;
if (devid_deviceid_to_nmlist("/dev/rdisk", devid,
    minor_name, &list))
    return (-1);
/* loop through list and process device names and numbers */
devid_free_nmlist(list);
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Interface Stability	Stable

SEE ALSO

`free(3C)`, `libdevid(3LIB)`, `attributes(5)`, `ddi_devid_compare(9F)`

NAME	di_binding_name, di_bus_addr, di_compatible_names, di_devid, di_driver_name, di_driver_ops, di_instance, di_nodeid, di_node_name – return libdevinfo node information
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> char *di_binding_name(di_node_t node); char *di_bus_addr(di_node_t node); int di_compatible_names(di_node_t node, char **names); ddi_devid_t di_devid(di_node_t node); char *di_driver_name(di_node_t node); uint_t di_driver_ops(di_node_t node); int di_instance(di_node_t node); int di_nodeid(di_node_t node); char *di_node_name(di_node_t node);</pre>
PARAMETERS	<i>names</i> The address of a pointer. <i>node</i> A handle to a device node.
DESCRIPTION	These functions extract information associated with a device node.
RETURN VALUES	The <code>di_binding_name()</code> function returns a pointer to the binding name. The binding name is the name used by the system to select a driver for the device. The <code>di_bus_addr()</code> function returns a pointer to a null-terminated string containing the assigned bus address for the device. NULL is returned if a bus address has not been assigned to the device. A zero-length string may be returned and is considered a valid bus address. The return value of <code>di_compatible_names()</code> is the number of compatible names. <i>names</i> is updated to point to a buffer contained within the snapshot. The buffer contains a concatenation of null-terminated strings, for example: <pre><name1>/0<name2>/0...<namen>/0</pre> See the discussion of generic names in <i>Writing Device Drivers</i> for a description of how compatible names are used by Solaris to achieve driver binding for the node. The <code>di_devid()</code> function returns the device ID for <i>node</i>, if it is registered. Otherwise, a null pointer is returned. Interfaces in the <code>libdevinfo(3LIB)</code> library may be used to manipulate the handle to the device id. This function is obsolete and may be removed from a future Solaris release. Applications should use the “devid” property instead.

di_binding_name(3DEVINFO)

The `di_driver_name()` function returns the name of the driver bound to the *node*. A null pointer is returned if *node* is not bound to any driver.

The `di_driver_ops()` function returns a bit array of device driver entry points that are supported by the driver bound to this *node*. Possible bit fields supported by the driver are `DI_CB_OPS`, `DI_BUS_OPS`, `DI_STREAM_OPS`.

The `di_instance()` function returns the instance number of the device. A value of -1 indicates an instance number has not been assigned to the device by the system.

The `di_nodeid()` function returns the type of device, which may be one of the following possible values: `DI_PSEUDO_NODEID`, `DI_PROM_NODEID`, and `DI_SID_NODEID`. Devices of type `DI_PROM_NODEID` may have additional properties that are defined by the PROM. See `di_prom_prop_data(3DEVINFO)` and `di_prom_prop_lookup_bytes(3DEVINFO)`.

The `di_node_name()` function returns a pointer to a null-terminated string containing the node name.

EXAMPLES See `di_init(3DEVINFO)` for an example demonstrating typical use of these functions.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving (<code>di_devid()</code> is obsolete)
MT-Level	Safe

SEE ALSO `di_init(3DEVINFO)`, `di_prom_init(3DEVINFO)`, `di_prom_prop_data(3DEVINFO)`, `di_prom_prop_lookup_bytes(3DEVINFO)`, `libdevid(3LIB)`, `libdevinfo(3DEVINFO)`, `attributes(5)`

Writing Device Drivers

di_child_node(3DEVINFO)

NAME	di_child_node, di_parent_node, di_sibling_node, di_drv_first_node, di_drv_next_node – libdevinfo node traversal functions						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_node_t di_child_node(di_node_t node); di_node_t di_parent_node(di_node_t node); di_node_t di_sibling_node(di_node_t node); di_node_t di_drv_first_node(const char *drv_name, di_node_t root); di_node_t di_drv_next_node(di_node_t node);</pre>						
PARAMETERS	<table><tr><td><i>drv_name</i></td><td>The name of the driver of interest.</td></tr><tr><td><i>node</i></td><td>A handle to any node in the snapshot.</td></tr><tr><td><i>root</i></td><td>The handle of the root node for the snapshot returned by <code>di_init()</code>.</td></tr></table>	<i>drv_name</i>	The name of the driver of interest.	<i>node</i>	A handle to any node in the snapshot.	<i>root</i>	The handle of the root node for the snapshot returned by <code>di_init()</code> .
<i>drv_name</i>	The name of the driver of interest.						
<i>node</i>	A handle to any node in the snapshot.						
<i>root</i>	The handle of the root node for the snapshot returned by <code>di_init()</code> .						
DESCRIPTION	The kernel device configuration data may be viewed in two ways, either as a tree of device configuration nodes or as a list of nodes associated with each driver. In the tree view, each node may contain references to its parent, the next sibling in a list of siblings, and the first child of a list of children. In the per-driver view, each node contains a reference to the next node associated with the same driver. Both views are captured in the snapshot, and the interfaces are provided for node access. The <code>di_child_node()</code> function obtains a handle to the first child of <i>node</i>. If no child node exists in the snapshot, <code>DI_NODE_NIL</code> is returned and <code>errno</code> is set to <code>ENXIO</code> or <code>ENOTSUP</code>. The <code>di_parent_node()</code> function obtains a handle to the parent node of <i>node</i>. If no parent node exists in the snapshot, <code>DI_NODE_NIL</code> is returned and <code>errno</code> is set to <code>ENXIO</code> or <code>ENOTSUP</code>. The <code>di_sibling_node()</code> function obtains a handle to the next sibling node of <i>node</i>. If no next sibling node exists in the snapshot, <code>DI_NODE_NIL</code> is returned and <code>errno</code> is set to <code>ENXIO</code> or <code>ENOTSUP</code>. The <code>di_drv_first_node()</code> function obtains a handle to the first node associated with the driver specified by <i>drv_name</i>. If there is no such driver, <code>DI_NODE_NIL</code> is returned with <code>errno</code> is set to <code>EINVAL</code>. If the driver exists but there is no node associated with this driver, <code>DI_NODE_NIL</code> is returned and <code>errno</code> is set to <code>ENXIO</code> or <code>ENOTSUP</code>. The <code>di_drv_next_node()</code> function returns a handle to the next node bound to the same driver. If no more nodes exist, <code>DI_NODE_NIL</code> is returned.						
RETURN VALUES	Upon successful completion, a handle is returned. Otherwise, <code>DI_NODE_NIL</code> is returned and <code>errno</code> is set to indicate the error.						

di_child_node(3DEVINFO)

ERRORS

These functions will fail if:

EINVAL	The argument is invalid.
ENXIO	The requested node does not exist.
ENOTSUP	The node was not found in the snapshot, but it may exist in the kernel. This error may occur if the snapshot contains a partial device tree.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO

di_init(), libdevinfo(3DEVINFO), attributes(5)

Writing Device Drivers

di_devfs_path(3DEVINFO)

NAME	di_devfs_path, di_devfs_path_free – generate and free physical path names						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> char *di_devfs_path(di_node_t node); void di_devfs_path_free(char *path_buf);</pre>						
PARAMETERS	<i>node</i> Handle to a device node in the snapshot. <i>path_buf</i> Pointer returned by di_devfs_path().						
DESCRIPTION	The di_devfs_path() function generates the physical path of the device <i>node</i>. The caller is responsible for freeing the memory allocated to store the physical path by calling di_devfs_path_free(). The di_devfs_path_free() function frees memory that was allocated by di_devfs_path().						
RETURN VALUES	The di_devfs_path() function returns a pointer to the string containing the physical path of <i>node</i> .						
ERRORS	The di_devfs_path() function will fail if: EINVAL The <i>node</i> argument is not a valid handle. The di_devfs_path() function can also return any error code from malloc(3C).						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	malloc(3C), libdevinfo(3DEVINFO), attributes(5) <i>Writing Device Drivers</i>						

di_init(3DEVINFO)

NAME	di_init, di_fini – create and destroy a snapshot of kernel device tree
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_node_t di_init(const char *phys_path, uint_t flags); void di_fini(di_node_t root);</pre>
PARAMETERS	<i>flags</i> Snapshot content specification. The possible values can be a bitwise OR of the following: - DINFOSUBTREE Include subtree. - DINFOPROP Include properties. - DINFOMINOR Include minor data. - DINFOCPYALL Include all of above. If <i>flags</i> is 0, the snapshot contains only a single node without properties or minor nodes. <i>phys_path</i> Physical path of the <i>root</i> node of the snapshot. See di_devfs_path(3DEVINFO). <i>root</i> Handle obtained by calling di_init().
DESCRIPTION	The di_init() function creates a snapshot of the kernel device tree and returns a handle of the <i>root</i> node. The caller specifies the contents of the snapshot by providing <i>flag</i> and <i>phys_path</i>. The di_fini() function destroys the snapshot of the kernel device tree and frees the associated memory. All handles associated with this snapshot become invalid after the call to di_fini().
RETURN VALUES	Upon success, di_init() returns a handle. Otherwise, DI_NODE_NIL is returned and errno is set to indicate the error.
ERRORS	The di_init() function can set errno to any error code that can also be set by open(2), ioctl(2) or mmap(2). The most common error codes include: - EACCESS Insufficient privilege for accessing device configuration data. - ENXIO Either the device named by <i>phys_path</i> is not present in the system, or the devinfo(7D) driver is not installed properly. - EINVAL Either <i>phys_path</i> is incorrectly formed or the <i>flags</i> argument is invalid.
EXAMPLES	EXAMPLE 1 Using the libdevinfo() Interfaces To Print All Device Tree Node Names The following is an example using the libdevinfo() interfaces to print all device tree node names: <pre>/* * Code to print all device tree node names */</pre>

EXAMPLE 1 Using the libdevinfo() Interfaces To Print All Device Tree Node Names
(Continued)

```

#include <stdio.h>
#include <libdevinfo.h>

int
prt_nodename(di_node_t node, void *arg)
{
    printf("%s\n", di_node_name(node));
    return (DI_WALK_CONTINUE);
}

main()
{
    di_node_t root_node;
    if((root_node = di_init("/", DINFOSUBTREE)) == DI_NODE_NIL) {
        fprintf(stderr, "di_init() failed\n");
        exit(1);
    }
    di_walk_node(root_node, DI_WALK_CLDFIRST, NULL, prt_nodename);
    di_fini(root_node);
}

```

EXAMPLE 2 Using the libdevinfo() Interfaces To Print The Physical Path Of SCSI Disks

The following example uses the libdevinfo() interfaces to print the physical path of SCSI disks:

```

/*
 * Code to print physical path of scsi disks
 */

#include <stdio.h>
#include <libdevinfo.h>
#define DISK_DRIVER "sd" /* driver name */

void
prt_diskinfo(di_node_t node)
{
    int instance;
    char *phys_path;

    /*
     * If the device node exports no minor nodes,
     * there is no physical disk.
     */
    if (di_minor_next(node, DI_MINOR_NIL) == DI_MINOR_NIL) {
        return;
    }

    instance = di_instance(node);
    phys_path = di_devfs_path(node);
    printf("%s%d: %s\n", DISK_DRIVER, instance, phys_path);
    di_devfs_path_free(phys_path);
}

```

di_init(3DEVINFO)

EXAMPLE 2 Using the libdevinfo() Interfaces To Print The Physical Path Of SCSI Disks *(Continued)*

```
}

void
walk_disknodes(di_node_t node)
{
    node = di_drv_first_node(DISK_DRIVER, node);
    while (node != DI_NODE_NIL) {
        prt_diskinfo(node);
        node = di_drv_next_node(node);
    }
}

main()
{
    di_node_t root_node;
    if ((root_node = di_init("/", DINFOCPYALL)) == DI_NODE_NIL) {
        fprintf(stderr, "di_init() failed\n");
        exit(1);
    }
    walk_disknodes(root_node);
    di_fini(root_node);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO open(2), ioctl(2), mmap(2), libdevinfo(3DEVINFO), attributes(5)

Writing Device Drivers

NAME	di_minor_devt, di_minor_name, di_minor_nodetype, di_minor_spectype – return libdevinfo minor node information						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> dev_t di_minor_devt(di_minor_t <i>minor</i>) ; char *di_minor_name(di_minor_t <i>minor</i>) ; char *di_minor_nodetype(di_minor_t <i>minor</i>) ; int di_minor_spectype(di_minor_t <i>minor</i>) ;</pre>						
PARAMETERS	<i>minor</i> A handle to minor data node.						
DESCRIPTION	These functions return libdevinfo minor node information.						
RETURN VALUES	The <code>di_minor_name()</code> function returns the <i>minor name</i>. See <code>ddi_create_minor_node(9F)</code> for a description of the <i>name</i> parameter. The <code>di_minor_devt()</code> function returns the <code>dev_t</code> value of the minor node that is specified by SYS V ABI. See <code>getmajor(9F)</code>, <code>getminor(9F)</code>, and <code>ddi_create_minor_node(9F)</code> for more information. The <code>di_minor_spectype()</code> function returns the <i>spec_type</i> of the file, either <code>S_IFCHR</code> or <code>S_IFBLK</code>. See <code>ddi_create_minor_node(9F)</code> for a description of the <i>spec_type</i> parameter. The <code>di_minor_nodetype()</code> function returns the <i>minor node_type</i> of the minor node. See <code>ddi_create_minor_node(9F)</code> for a description of the <i>node_type</i> parameter.						
ERRORS	No errors are defined.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	<code>attributes(5)</code>, <code>ddi_create_minor_node(9F)</code>, <code>getmajor(9F)</code>, <code>getminor(9F)</code> <i>Writing Device Drivers</i>						

di_minor_next(3DEVINFO)

NAME	di_minor_next – libdevinfo minor node traversal functions						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_minor_t di_minor_next(di_node_t node, di_minor_t minor);</pre>						
PARAMETERS	<i>minor</i> Handle to the current minor node or DI_MINOR_NIL. <i>node</i> Device node with which the minor node is associated.						
DESCRIPTION	The <code>di_minor_next()</code> function returns a handle to the next minor node for the device node <i>node</i> . If <i>minor</i> is DI_MINOR_NIL, a handle to the first minor node is returned.						
RETURN VALUES	Upon successful completion, a handle to the next minor node is returned. Otherwise, DI_MINOR_NIL is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>di_minor_next()</code> function will fail if: EINVAL Invalid argument. ENOTSUP Minor node information is not available in snapshot. ENXIO End of minor node list.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	<code>libdevinfo(3DEVINFO)</code> , <code>attributes(5)</code> <i>Writing Device Drivers</i>						

di_prom_init(3DEVINFO)

NAME	di_prom_init, di_prom_fini – create and destroy a handle to the PROM device information						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_prom_handle_t di_prom_init(void); void di_prom_fini(di_prom_handle_t ph);</pre>						
PARAMETERS	<i>ph</i> Handle to prom returned by di_prom_init().						
DESCRIPTION	For device nodes whose nodeid value is DI_PROM_NODEID (see di_nodeid(3DEVINFO)), additional properties can be retrieved from the PROM. The di_prom_init() function returns a handle that is used to retrieve such properties. This handle is passed to di_prom_prop_lookup_bytes(3DEVINFO) and di_prom_prop_next(3DEVINFO). The di_prom_fini() function destroys the handle and all handles to the PROM device information obtained from that handle.						
RETURN VALUES	Upon successful completion, di_prom_init() returns a handle. Otherwise, DI_PROM_HANDLE_NIL is returned and errno is set to indicate the error.						
ERRORS	The di_prom_init() sets errno function to any error code that can also be set by openprom(7D) or malloc(3C).						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	di_nodeid(3DEVINFO), di_prom_prop_next(3DEVINFO), di_prom_prop_lookup_bytes(3DEVINFO), libdevinfo(3DEVINFO), malloc(3C), attributes(5), openprom(7D)						

di_prom_prop_data(3DEVINFO)

NAME	di_prom_prop_data, di_prom_prop_next, di_prom_prop_name – access PROM device information
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_prom_prop_t di_prom_prop_next(di_prom_handle_t <i>ph</i>, di_node_t <i>node</i>, di_prom_prop_t <i>prom_prop</i>) ; char *di_prom_prop_name(di_prom_prop_t <i>prom_prop</i>) ; int di_prom_prop_data(di_prom_prop_t <i>prom_prop</i>, uchar_t **<i>prop_data</i>) ;</pre>
PARAMETERS	<i>node</i> Handle to a device node in the snapshot of kernel device tree. <i>ph</i> PROM handle <i>prom_prop</i> Handle to a PROM property. <i>prop_data</i> Address of a pointer.
DESCRIPTION	The <code>di_prom_prop_next()</code> function obtains a handle to the next property on the PROM property list associated with <i>node</i>. If <i>prom_prop</i> is <code>DI_PROM_PROP_NIL</code>, the first property associated with <i>node</i> is returned. The <code>di_prom_prop_name()</code> function returns the name of the <i>prom_prop</i> property. The <code>di_prom_prop_data()</code> function returns the value of the <i>prom_prop</i> property. The return value is a non-negative integer specifying the size in number of bytes in <i>prop_data</i>. All memory allocated by these functions is managed by the library and must not be freed by the caller.
RETURN VALUES	The <code>di_prom_prop_data()</code> function returns the number of bytes in <i>prop_data</i> and <i>prop_data</i> is updated to point to a byte array containing the property value. If 0 is returned, the property is a boolean property and the existence of this property indicates the value is true. The <code>di_prom_prop_name()</code> function returns a pointer to a string that contains the name of <i>prom_prop</i>. The <code>di_prom_prop_next()</code> function returns a handle to the next PROM property. <code>DI_PROM_PROP_NIL</code> is returned if no additional properties exist.
ERRORS	See <code>openprom(7D)</code> for a description of possible errors.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

di_prom_prop_data(3DEVINFO)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO attributes(5), openprom(7D)

Writing Device Drivers

di_prom_prop_lookup_bytes(3DEVINFO)

NAME	di_prom_prop_lookup_bytes, di_prom_prop_lookup_ints, di_prom_prop_lookup_strings – search for a PROM property
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> int di_prom_prop_lookup_bytes(di_prom_handle_t ph, di_node_t node, const char *prop_name, uchar_t **prop_data); int di_prom_prop_lookup_ints(di_prom_handle_t ph, di_node_t node, const char *prop_name, int **prop_data); int di_prom_prop_lookup_strings(di_prom_handle_t ph, di_node_t node, const char *prop_name, char **prop_data);</pre>
PARAMETERS	<i>node</i> Handle to device node in snapshot created by di_init(3DEVINFO). <i>ph</i> Handle returned by di_prom_init(3DEVINFO). <i>prop_data</i> For di_prom_prop_lookup_bytes(), the address of a pointer to an array of unsigned characters. For di_prom_prop_lookup_ints(), the address of a pointer to an integer. For di_prom_prop_lookup_strings(), the address of pointer to a buffer. <i>prop_name</i> The name of the property being searched.
DESCRIPTION	These functions return the value of a known PROM property name and value type and update the <i>prop_data</i> pointer to reference memory that contains the property value. All memory allocated by these functions is managed by the library and must not be freed by the caller.
RETURN VALUES	If the property is found, the number of entries in <i>prop_data</i> is returned. If the property is a boolean type, 0 is returned and the existence of this property indicates the value is true. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. For di_prom_prop_lookup_bytes(), the number of entries is the number of unsigned characters contained in the buffer pointed to by <i>prop_data</i>. For di_prom_prop_lookup_ints(), the number of entries is the number of integers contained in the buffer pointed to by <i>prop_data</i>. For di_prom_prop_lookup_strings(), the number of entries is the number of null-terminated strings contained in the buffer. The strings are stored in a concatenated format in the buffer.
ERRORS	These functions will fail if: EINVAL Invalid argument.

di_prom_prop_lookup_bytes(3DEVINFO)

ENXIO

The property does not exist.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO

di_init(3DEVINFO), di_prom_prop_next(3DEVINFO),
libdevinfo(3DEVINFO), attributes(5), openprom(7D)

Writing Device Drivers

di_prop_bytes(3DEVINFO)

NAME	di_prop_bytes, di_prop_devt, di_prop_ints, di_prop_name, di_prop_strings, di_prop_type, di_prop_int64 – access property values and attributes
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> int di_prop_bytes(di_prop_t prop, uchar_t **prop_data); dev_t di_prop_devt(di_prop_t prop); int di_prop_ints(di_prop_t prop, int **prop_data); int di_prop_int64(di_prop_t prop, int64_t **prop_data); char *di_prop_name(di_prop_t prop); int di_prop_strings(di_prop_t prop, char **prop_data); int di_prop_type(di_prop_t prop);</pre>
PARAMETERS	<i>prop</i> Handle to a property returned by di_prop_next(3DEVINFO). <i>prop_data</i> For di_prop_bytes(), the address of a pointer to an unsigned character. For di_prop_ints(), the address of a pointer to an integer. For di_prop_int64(), the address of a pointer to a 64-bit integer. For di_prop_strings(), the address of pointer to a character.
DESCRIPTION	These functions access information associated with property values and attributes. All memory allocated by these functions is managed by the library and must not be freed by the caller. The di_prop_bytes() function returns the property data as a series of unsigned characters. The di_prop_devt() function returns the dev_t with which this property is associated. If the value is DDI_DEV_T_NONE, the property is not associated with any specific minor node. The di_prop_ints() function returns the property data as a series of integers. The di_prop_int64() function returns the property data as a series of 64-bit integers. The di_prop_name() function returns the name of the property. The di_prop_strings() function returns the property data as a concatenation of null-terminated strings.

di_prop_bytes(3DEVINFO)

The `di_prop_type()` function returns the type of the property. The type determines the appropriate interface to access property values. The following is a list of possible types:

<code>DI_PROP_TYPE_BOOLEAN</code>	There is no interface to call since there is no property data associated with boolean properties. The existence of the property defines a TRUE value.
<code>DI_PROP_TYPE_INT</code>	Use <code>di_prop_ints()</code> to access property data.
<code>DI_PROP_TYPE_INT64</code>	Use <code>di_prop_int64()</code> to access property data.
<code>DI_PROP_TYPE_STRING</code>	Use <code>di_prop_strings()</code> to access property data.
<code>DI_PROP_TYPE_BYTE</code>	Use <code>di_prop_bytes()</code> to access property data.
<code>DI_PROP_TYPE_UNKNOWN</code>	Use <code>di_prop_bytes()</code> to access property data. Since the type of property is unknown, the caller is responsible for interpreting the contents of the data.
<code>DI_PROP_TYPE_UNDEF_IT</code>	The property has been undefined by the driver. No property data is available.

RETURN VALUES

Upon successful completion, `di_prop_bytes()`, `di_prop_ints()`, `di_prop_int64()`, and `di_prop_strings()` return a non-negative value, indicating the number of entries in the property value buffer. See `di_prom_prop_lookup_bytes(3DEVINFO)` for a description of the return values. Otherwise, -1 is returned and `errno` is set to indicate the error.

The `di_prop_devt()` function returns the `dev_t` value associated with the property.

The `di_prop_name()` function returns a pointer to a string containing the name of the property.

The `di_prop_type()` function can return one of types described in the DESCRIPTION section.

ERRORS

These functions will fail if:

<code>EINVAL</code>	Invalid argument. For example, the property type does not match the interface.

ATTRIBUTES

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

di_prop_bytes(3DEVINFO)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO `di_prom_prop_lookup_bytes(3DEVINFO)`, `di_prop_next(3DEVINFO)`,
`libdevinfo(3DEVINFO)`, `attributes(5)`

Writing Device Drivers

NAME	di_prop_lookup_bytes, di_prop_lookup_ints, di_prop_lookup_int64, di_prop_lookup_strings – search for a property
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> int di_prop_lookup_bytes(dev_t dev, di_node_t node, const char *prop_name, uchar_t **prop_data); int di_prop_lookup_ints(dev_t dev, di_node_t node, const char *prop_name, int **prop_data); int di_prop_lookup_int64(dev_t dev, di_node_t node, const char *prop_name, int64_t **prop_data); int di_prop_lookup_strings(dev_t dev, di_node_t node, const char *prop_name, char **prop_data);</pre>
PARAMETERS	<i>dev</i> dev_t of minor node with which the property is associated. DDI_DEV_T_ANY is a wild card that matches all dev_t's, including DDI_DEV_T_NONE. <i>node</i> Handle to the device node with which the property is associated. <i>prop_data</i> For di_prop_lookup_bytes(), the address to a pointer to an array of unsigned characters containing the property data. For di_prop_lookup_ints(), the address to a pointer to an array of integers containing the property data. For di_prop_lookup_int64(), the address to a pointer to an array of 64-bit integers containing the property data. For di_prop_lookup_strings(), the address to a pointer to a buffer containing a concatenation of null-terminated strings containing the property data. <i>prop_name</i> Name of the property for which to search.
DESCRIPTION	These functions return the value of a known property name type and dev_t value. All memory allocated by these functions is managed by the library and must not be freed by the caller.
RETURN VALUES	If the property is found, the number of entries in <i>prop_data</i> is returned. If the property is a boolean type, 0 is returned and the existence of this property indicates the value is true. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error.
ERRORS	These functions will fail if: EINVAL Invalid argument. ENOTSUP The snapshot contains no property information. ENXIO The property does not exist; try di_prom_prop_lookup_*().

di_prop_lookup_bytes(3DEVINFO)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Safe

SEE ALSO `di_init(3DEVINFO)`, `di_prom_prop_lookup_bytes(3DEVINFO)`,
`libdevinfo(3DEVINFO)`, `attributes(5)`

Writing Device Drivers

NAME	di_prop_next – libdevinfo property traversal function						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> di_prop_t di_prop_next(di_node_t node, di_prop_t prop);</pre>						
PARAMETERS	<i>node</i> Handle to a device node. <i>prop</i> Handle to a property.						
DESCRIPTION	The di_prop_next () function returns a handle to the next property on the property list. If <i>prop</i> is DI_PROP_NIL, the handle to the first property is returned.						
RETURN VALUES	Upon successful completion, di_prop_next () returns a handle. Otherwise DI_PROP_NIL is returned and errno is set to indicate the error.						
ERRORS	The di_prop_next () function will fail if: EINVAL Invalid argument. ENOTSUP The snapshot does not contain property information. ENXIO There are no more properties.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	di_init(3DEVINFO), libdevinfo(3DEVINFO), attributes(5) <i>Writing Device Drivers</i>						

DisconnectToServer(3DMI)

NAME	DisconnectToServer – disconnect from a DMI service provider
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmici -ldmimi [<i>library</i> ...] #include <dmide/api.hh> bool_t DisconnectToServer(DmiRpcHandle *<i>dmi_rpc_handle</i>) ;</pre>
DESCRIPTION	The <code>DisconnectToServer()</code> function disconnects a management application or a component instrumentation from a DMI service provider.
RETURN VALUES	The <code>ConnectToServer()</code> function returns TRUE if successful, otherwise FALSE.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	Safe

SEE ALSO `ConnectToServer(3DMI)`, `attributes(5)`

NAME	di_walk_minor – traverse libdevinfo minor nodes						
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> int di_walk_minor(di_node_t root, const char *minor_nodetype, uint_t flag, void *arg, int (*minor_callback)di_node_t node, di_minor_t minor, void *arg);</pre>						
PARAMETERS	<i>arg</i> Pointer to caller– specific user data. <i>flag</i> Specify 0. Reserved for future use. <i>minor</i> The minor node visited. <i>minor_nodetype</i> A character string specifying the minor data type, which may be one of the types defined by the Solaris DDI framework, for example, DDI_NT_BLOCK. NULL matches all <i>minor_node</i> types. See ddi_create_minor_node(9F). <i>node</i> The device node with which the minor node is associated. <i>root</i> Root of subtree to visit.						
DESCRIPTION	The di_walk_minor() function visits all minor nodes attached to device nodes in a subtree rooted at <i>root</i> . For each minor node that matches <i>minor_nodetype</i> , the caller-supplied function <i>minor_callback</i> () is invoked. The walk terminates immediately when <i>minor_callback</i> () returns DI_WALK_TERMINATE.						
RETURN VALUES	Upon successful completion, di_walk_minor() returns 0. Otherwise, -1 is returned and errno is set to indicate the error. The minor_callback() function returns one of the following: DI_WALK_CONTINUE Continue to visit subsequent minor data nodes. DI_WALK_TERMINATE Terminate the walk immediately.						
ERRORS	The di_walk_minor() function will fail if: EINVAL Invalid argument.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	di_minor_nodetype(3DEVINFO), libdevinfo(3DEVINFO), attributes(5), ddi_create_minor_node(9F)						

di_walk_minor(3DEVINFO)

Writing Device Drivers

NAME	di_walk_node – traverse libdevinfo device nodes								
SYNOPSIS	<pre>cc [flag...] file... -ldevinfo [library...] #include <libdevinfo.h> int di_walk_node(di_node_t root, uint_t flag, void *arg, int (*node_callback)di_node_t node, void *arg);</pre>								
DESCRIPTION	The <code>di_walk_node()</code> function visits all nodes in the subtree rooted at <i>root</i> . For each node found, the caller-supplied function <i>node_callback()</i> is invoked. The return value of <i>node_callback()</i> specifies subsequent walking behavior.								
PARAMETERS	<i>arg</i> Pointer to caller-specific data. <i>flag</i> Specifies walking order, either <code>DI_WALK_CLDFIRST</code> (depth first) or <code>DI_WALK_SIBFIRST</code> (breadth first). <code>DI_WALK_CLDFIRST</code> is the default. <i>node</i> The node being visited. <i>root</i> The handle to the root node of the subtree to visit.								
RETURN VALUES	Upon successful completion, <code>di_walk_node()</code> returns 0. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. The <i>node_callback()</i> function can return one of the following: <table> <tr> <td><code>DI_WALK_CONTINUE</code></td><td>Continue walking.</td></tr> <tr> <td><code>DI_WALK_PRUNESIB</code></td><td>Continue walking, but skip siblings and their child nodes.</td></tr> <tr> <td><code>DI_WALK_PRUNECILD</code></td><td>Continue walking, but skip subtree rooted at current node .</td></tr> <tr> <td><code>DI_WALK_TERMINATE</code></td><td>Terminate the walk immediately.</td></tr> </table>	<code>DI_WALK_CONTINUE</code>	Continue walking.	<code>DI_WALK_PRUNESIB</code>	Continue walking, but skip siblings and their child nodes.	<code>DI_WALK_PRUNECILD</code>	Continue walking, but skip subtree rooted at current node .	<code>DI_WALK_TERMINATE</code>	Terminate the walk immediately.
<code>DI_WALK_CONTINUE</code>	Continue walking.								
<code>DI_WALK_PRUNESIB</code>	Continue walking, but skip siblings and their child nodes.								
<code>DI_WALK_PRUNECILD</code>	Continue walking, but skip subtree rooted at current node .								
<code>DI_WALK_TERMINATE</code>	Terminate the walk immediately.								
ERRORS	The <code>di_walk_node()</code> function will fail if: <table> <tr> <td><code>EINVAL</code></td><td>Invalid argument.</td></tr> </table>	<code>EINVAL</code>	Invalid argument.						
<code>EINVAL</code>	Invalid argument.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Evolving								
MT-Level	Safe								
SEE ALSO	<code>di_init(3DEVINFO)</code>, <code>libdevinfo(3DEVINFO)</code>, <code>attributes(5)</code> <i>Writing Device Drivers</i>								

DmiAddComponent(3DMI)

NAME	DmiAddComponent, DmiAddGroup, DmiAddLanguage, DmiDeleteComponent, DmiDeleteGroup, DmiDeleteLanguage – Management Interface database administration functions
SYNOPSIS	<pre>cc [flag ...] file ... -ldmimi -ldmi -lnsl -lrwtool [library ...] #include <dmi/server.h> #include <dmi/miapi.h> bool_t DmiAddComponent(DmiAddComponentIN <i>argin</i>, DmiAddComponentOUT *result, DmiRpcHandle *dmi_rpc_handle); bool_t DmiAddGroup(DmiAddGroupIN <i>argin</i>, DmiAddGroupOUT *result, DmiRpcHandle *dmi_rpc_handle); bool_t DmiAddLanguage(DmiAddLanguageIN <i>argin</i>, DmiAddLanguageOUT *result, DmiRpcHandle *dmi_rpc_handle); bool_t DmiDeleteComponent(DmiDeleteComponentIN <i>argin</i>, DmiDeleteComponentOUT *result, DmiRpcHandle *dmi_rpc_handle); bool_t DmiDeleteGroup(DmiDeleteGroupIN <i>argin</i>, DmiDeleteGroupOUT *result, DmiRpcHandle *dmi_rpc_handle); bool_t DmiDeleteLanguage(DmiDeleteLanguageIN <i>argin</i>, DmiDeleteLanguageOUT *result, DmiRpcHandle *dmi_rpc_handle);</pre>
DESCRIPTION	The database administration functions add a new component to the database or add a new language mapping for an existing component. You may also remove an existing component, remove a specific language mapping, or remove a group from a component. The <code>DmiAddComponent()</code> function adds a new component to the DMI database. It takes the name of a file, or the address of memory block containing MIF data, checks the data for adherence to the DMI MIF grammar, and installs the MIF in the database. The procedure returns a unique component ID for the newly installed component. The <i>argin</i> parameter is an instance of a <code>DmiAddComponentIN</code> structure containing the following members: <pre>DmiHandle_t handle; /* an open session handle */ DmiFileDataList_t *fileData; /* MIF data for component */</pre> The <i>result</i> parameter is a pointer to a <code>DmiAddComponentOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status; DmiId_t compId; /* SP-allocated component ID */ DmiStringList_t *errors; /* installation error messages */</pre> The <code>DmiAddLanguage()</code> function adds a new language mapping for an existing component in the database. It takes the name of a file, or the address of memory block containing translated MIF data, checks the data for adherence to the DMI MIF grammar, and installs the language MIF in the database. The <i>argin</i> parameter is an instance of a <code>DmiAddLanguageIN</code> structure containing the following members:

DmiAddComponent(3DMI)

```
DmiHandle_t      handle;          /* an open session handle */
DmiFileDataList_t *fileData;      /* language mapping file */
DmiId_t          compId;          /* component to access */
```

The *result* parameter is a pointer to a DmiAddLanguageOUT structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiStringList_t  *errors;        /* installation error messages */
```

The DmiAddGroup() function adds a new group to an existing component in the database. It takes the name of a file, or the address of memory block containing the group's MIF data, checks the data for adherence to the DMI MIF grammar, and installs the group MIF in the database. The *argIn* parameter is an instance of a DmiAddGroupIN structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiFileDataList_t *fileData;      /* MIF file data for group */
DmiId_t          compId;          /* component to access */
```

The *result* parameter is a pointer to a DmiAddGroupOUT structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiId_t          groupId;         /* SP-allocated group ID */
DmiStringList_t  *errors;        /* installation error messages */
```

The DmiDeleteComponent() function removes an existing component from the database. The *argIn* parameter is an instance of a DmiDeleteComponentIN structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiId_t          compId;          /* component to delete */
```

The *result* parameter is a pointer to a DmiDeleteComponentOUT structure containing the following members:

```
DmiErrorStatus_t error_status;
```

The DmiDeleteLanguage() function removes a specific language mapping for a component. You specify the language string and component ID. The *argIn* parameter is an instance of a DmiDeleteLanguageIN structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiString_t      *language;       /* language to delete */
DmiId_t          compId;          /* component to access */
```

The *result* parameter is a pointer to a DmiDeleteLanguageOUT structure containing the following members:

```
DmiErrorStatus_t error_status;
```

The DmiDeleteGroup() function removes a group from a component. The caller specifies the component and group IDs. The *argIn* parameter is an instance of a DmiDeleteGroupIN structure containing the following members:

DmiAddComponent(3DMI)

```
DmiHandle_t      handle;          /* an open session handle */
DmiId_t          compId;          /* component containing group */
DmiId_t          groupId;         /* group to delete */
```

The *result* parameter is a pointer to a DmiDeleteGroupOUT structure containing the following members:

```
DmiErrorStatus_t  error_status;
```

RETURN VALUES

The DmiAddComponent () function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE
```

The DmiAddGroup () function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE
```

The DmiAddLanguage () function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE
```

The DmiDeleteComponent () function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The DmiDeleteGroup () function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
```


DmiAddComponent(3DMI)

DMIERR_SP_INACTIVE
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR

The DmiDeleteLanguage() function returns the following possible values:

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWsasdk
MT-level	Unsafe

SEE ALSO attributes(5)

DmiAddRow(3DMI)

NAME	DmiAddRow, DmiDeleteRow, DmiGetAttribute, DmiGetMultiple, DmiSetAttribute, DmiSetMultiple – Management Interface operation functions
SYNOPSIS	<pre>cc [flag ...] file ... -ldmimi -ldmi -lnsl -lrwtool [library ...] #include <server.h> #include <miapi.h> bool_t DmiAddRow(DmiAddRowIN <i>argin</i>, DmiAddRowOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiDeleteRow(DmiDeleteRowIN <i>argin</i>, DmiDeleteRowOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiGetAttribute(DmiGetAttributeIN <i>argin</i>, DmiGetAttributeOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiGetMultiple(DmiGetMultipleIN <i>argin</i>, DmiGetMultipleOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiSetAttribute(DmiSetAttributeIN <i>argin</i>, DmiSetAttributeOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiSetMultiple(DmiSetMultipleIN <i>argin</i>, DmiSetMultipleOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>);</pre>
DESCRIPTION	The operation functions provide a method for retrieving a single value from the Service Provider and for setting a single attribute value. In addition, you may also retrieve attribute values from the Service Provider. You may perform a set operation on an attribute or a list of attributes and add or delete a row from an existing table. The <code>DmiAddRow()</code> function adds a row to an existing table. The <code>rowData</code> parameter contains the full data, including key attribute values, for a row. It is an error for the key list to specify an existing table row. The <i>argin</i> parameter is an instance of a <code>DmiAddRowIN</code> structure containing the following members: <pre>DmiHandle_t handle; /* An open session handle */ DmiRowData_t *rowData; /* Attribute values to set */</pre> The <i>result</i> parameter is a pointer to a <code>DmiAddRowOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status;</pre> <code>DmiDeleteRow()</code> function removes a row from an existing table. The key list must specify valid keys for a table row. The <i>argin</i> parameter is an instance of a <code>DmiDeleteRowIN</code> structure containing the following members: <pre>DmiHandle_t handle; /* An open session handle */ DmiRowData_t *rowData; /* Row to delete */</pre> The <i>result</i> parameter is a pointer to a <code>DmiDeleteRowOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status;</pre>

The `DmiGetAttribute()` function provides a simple method for retrieving a single attribute value from the Service Provider. The `compId`, `groupId`, `attribId`, and `keyList` identify the desired attribute. The resulting attribute value is returned in a newly allocated `DmiDataUnion` structure. The address of this structure is returned through the `value` parameter. The *argIn* parameter is an instance of a `DmiListComponentsIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiId_t          compId;          /* Component to access */
DmiId_t          groupId;         /* Group within component */
DmiId_t          attribId;        /* Attribute within a group */
DmiAttributeValues_t *keyList;    /* Keylist to specify a table row */
```

The *result* parameter is a pointer to a `DmiGetAttributeOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiDataUnion_t   *value;        /* Attribute value returned */
```

The `DmiGetMultiple()` function retrieves attribute values from the Service Provider. This procedure may get the value for an individual attribute, or for multiple attributes across groups, components, or rows of a table.

The `DmiSetAttribute()` function provides a simple method for setting a single attribute value. The `compId`, `groupId`, `attribId`, and `keyList` identify the desired attribute. The `setMode` parameter defines the procedure call as a Set, Reserve, or Release operation. The new attribute value is contained in the `DmiDataUnion` structure whose address is passed in the `value` parameter. The *argIn* parameter is an instance of a `DmiSetAttributeIN` structure containing the following members:

```
DmiHandle_t      handle;
DmiId_t          compId;
DmiId_t          groupId;
DmiId_t          attribId;
DmiAttributeValues_t *keyList;
DmiSetMode_t     setMode;
DmiDataUnion_t   *value;
```

The *result* parameter is a pointer to a `DmiSetAttributeOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```

The `DmiSetMultiple()` function performs a set operation on an attribute or list of attributes. Set operations include actually setting the value, testing and reserving the attribute for future setting, or releasing the set reserve. These variations on the set operation are specified by the parameter `setMode`. The *argIn* parameter is an instance of a `DmiSetMultipleIN` structure containing the following members:

```
DmiHandle_t      handle;          /* An open session handle */
DmiSetMode_t     setMode;         /* set, reserve, or release */
DmiMultiRowData_t *rowData;       /* Attribute values to set */
```

The *result* parameter is a pointer to a `DmiSetMultipleOUT` structure containing the following members:

DmiAddRow(3DMI)

```
DmiErrorStatus_t      error_status;
```

The `rowData` array describes the attributes to set, and contains the new attribute values. Each element of `rowData` specifies a component, group, key list (for table accesses), and attribute list to set. No data is returned from this function.

RETURN VALUES

The `DmiAddRow()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_VALUE_UNKNOWN
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_VALUE_UNKNOWN
DMIERR_UNABLE_TO_ADD_ROW
```

The `DmiDeleteRow()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_VALUE_UNKNOWN
DMIERR_UNABLE_TO_DELETE_ROW
```

The `DmiGetAttribute()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN
```

The `DmiGetMultiple()` function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_RPC_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

The `DmiSetAttribute()` function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

The `DmiSetMultiple()` function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_SET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

DmiAddRow(3DMI)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	Unsafe

SEE ALSO `attributes(5)`

NAME	dmi_error – print error in string form				
SYNOPSIS	<pre>cc [flag ...] file ... -ldmi -lnsl -lrwtool [library ...] #include <dmi/dmi_error.hh> void dmi_error(DmiErrorStatus_t error_status);</pre>				
DESCRIPTION	For the given <i>error_status</i> , the <code>dmi_error()</code> function prints the corresponding error in string form. The function prints "unknown dmi errors" if <i>error_status</i> is invalid.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-level	MT-Safe				
SEE ALSO	<code>libdmi(3LIB)</code> , <code>attributes(5)</code>				

DmiGetConfig(3DMI)

NAME	DmiGetConfig, DmiGetVersion, DmiRegister, DmiSetConfig, DmiUnregister – Management Interface initialization functions
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmimi -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <miapi.h> bool_t DmiGetConfig(DmiGetConfigIN <i>argin</i>, DmiGetConfigOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiGetVersion(DmiGetVersionIN <i>argin</i>, DmiGetVersionOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiRegister(DmiRegisterIN <i>argin</i>, DmiRegisterOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiSetConfig(DmiSetConfigIN <i>argin</i>, DmiSetConfigOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>); bool_t DmiUnregister(DmiUnregisterIN <i>argin</i>, DmiUnregisterOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>);</pre>
DESCRIPTION	The Management Interface initialization functions enable you to register management applications to the Service Provider. You may also retrieve information about the Service Provider, get and set session configuration information for your session. The <code>DmiGetConfig()</code> function retrieves the per-session configuration information. The configuration information consists of a string describing the current language being used for the session. The <i>argin</i> parameter is an instance of a <code>DmiGetConfigIN</code> structure containing the following member: <pre>DmiHandle_t handle; /* an open session handle */</pre> The <i>result</i> parameter is a pointer to a <code>DmiGetConfigOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status; DmiString_t *language; /* current session language */</pre> The <code>DmiGetVersion()</code> function retrieves information about the Service Provider. The management application uses the <code>DmiGetVersion()</code> procedure to determine the DMI specification level supported by the Service Provider. This procedure also returns the service provided description string, and may contain version information about the Service Provider implementation. The <i>argin</i> parameter is an instance of a <code>DmiGetVersionIN</code> structure containing the following member: <pre>DmiHandle_t handle; /* an open session handle */</pre> The <i>result</i> parameter is a pointer to a <code>DmiGetVersionOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status; DmiString_t *dmiSpecLevel; /* DMI specification version */ DmiString_t *description; /* OS specific DMI SP version */ DmiFileTypeList_t *fileTypes; /* file types for MIF installation */</pre>

DmiGetConfig(3DMI)

The `DmiRegister()` function provides the management application with a unique per-session handle. The Service Provider uses this procedure to initialize to an internal state for subsequent procedure calls made by the application. This procedure must be the first command executed by the management application. *argIn* is an instance of a `DmiRegisterIN` structure containing the following member:

```
DmiHandle_t      handle;          /* an open session handle */
```

The *result* parameter is a pointer to a `DmiRegisterOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;  
DmiHandle_t      *handle;        /* an open session handle */
```

The `DmiSetConfig()` function sets the per-session configuration information. The configuration information consists of a string describing the language required by the management application. The *argIn* parameter is an instance of a `DmiSetConfigIN` structure containing the following member:

```
DmiHandle_t      handle;          /* an open session handle */  
DmiString_t      *language;      /* current language required */
```

The *result* parameter is a pointer to a `DmiSetConfigOUT` structure containing the following member:

```
DmiErrorStatus_t error_status;
```

The `DmiUnregister()` function is used by the Service Provider to perform end-of-session cleanup actions. On return from this function, the session handle is no longer valid. This function must be the last DMI command executed by the management application. The *argIn* parameter is an instance of a `DmiUnregisterIN` structure containing the following member:

```
DmiHandle_t      handle;          /* an open session handle */
```

The *result* parameter is a pointer to a `DmiUnregisterOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```

RETURN VALUES

The `DmiGetConfig()` function returns the following possible values:

```
DMIERR_NO_ERROR  
DMIERR_ILLEGAL_RPC_HANDLE  
DMIERR_OUT_OF_MEMORY  
DMIERR_ILLEGAL_PARAMETER  
DMIERR_SP_INACTIVE
```

The `DmiGetVersion()` function returns the following possible values:

```
DMIERR_NO_ERROR  
DMIERR_ILLEGAL_RPC_HANDLE  
DMIERR_OUT_OF_MEMORY  
DMIERR_SP_INACTIVE
```

The `DmiRegister()` function returns the following possible values:

DmiGetConfig(3DMI)

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_SP_INACTIVE

The DmiSetConfig () function returns the following possible values:

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ILLEGAL_TO_SET

The DmiUnRegister () function returns the following possible values:

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	Unsafe

SEE ALSO

attributes(5)

NAME	DmiListAttributes, DmiListClassNames, DmiListComponents, DmiListComponentsByClass, DmiListGroups, DmiListLanguages – Management Interface listing functions
SYNOPSIS	<pre>cc [flag ...] file ... -ldmimi -ldmi -lnsl -lrwtool [library ...] #include <server.h> #include <miapi.h> bool_t DmiListAttributes(DmiListAttributesIN <i>argin</i>, DmiListAttributesOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiListClassNames(DmiListClassNamesIN <i>argin</i>, DmiListClassNamesOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiListComponents(DmiListComponentsIN <i>argin</i>, DmiListComponentsOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiListComponentsByClass(DmiListComponentsByClassIN <i>argin</i>, DmiListComponentsByClassOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiListGroups(DmiListGroupsIN <i>argin</i>, DmiListGroupsOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiListLanguages(DmiListLanguagesIN <i>argin</i>, DmiListLanguagesOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ;</pre>
DESCRIPTION	The listing functions enables you to retrieve the names and the description of components in a system. You may also list components by class that match a specified criteria. The listing functions retrieve the set of language mappings installed for a specified component, retrieve class name strings for all groups in a component, retrieve a list of groups within a component, and retrieve the properties for one or more attributes in a group. The <code>DmiListComponents()</code> function retrieves the name and (optionally) the description of components in a system. Use this to interrogate a system to determine what components are installed. The <i>argin</i> parameter is an instance of a <code>DmiListComponentsIN</code> structure containing the following members: <pre>DmiHandle_t handle; /* an open session handle */ DmiRequestMode_t requestMode; /* Unique, first, or next */ DmiUnsigned_t maxCount; /* maximum number to return, 0 for all */ DmiBoolean_t getPragma; /* get optional pragma string */ DmiBoolean_t getDescription; /* get optional component description */ DmiId_t compId; /* component ID to start with */</pre> The <i>result</i> parameter is a pointer to a <code>DmiListComponentsOUT</code> structure containing the following members: <pre>DmiErrorStatus_t error_status; DmiComponentList_t *reply; /* list of components */</pre>

DmiListAttributes(3DMI)

An enumeration accesses a specific component or may be used to sequentially access all components in a system. The caller may choose not to retrieve the component description by setting the value `getDescription` to false. The caller may choose not to retrieve the pragma string by setting the value of `gutta-percha` to false. The `maxCount`, `requestMode`, and `compId` parameters allow the caller to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `compId` specifies the first component requested (or only component if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `compId` specifies the component just before the one requested. When `requestMode` is `DMI_FIRST`, `compId` is unused.

To control the amount of information returned, the caller sets `maxCount` to something other than zero. The service provider must honor this limit on the amount of information returned. When `maxCount` is 0 the service provider returns information for all components, subject to the constraints imposed by `requestMode` and `compId`.

The `DmiListComponentsByClass()` function lists components that match specified criteria. Use this function to determine if a component contains a certain group or a certain row in a table. A filter condition may be that a component contains a specified group class name or that it contains a specific row in a specific group. As with `DmiListComponents()`, the description and pragma strings are optional return values. *argIn* is an instance of a `DmiListComponentsByClassIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiRequestMode_t requestMode;     /* Unique, first or next */
DmiUnsigned_t    maxCount;        /* maximum number to return,
                                   or 0 for all */
DmiBoolean_t     getPragma;       /* get the optional pragma
                                   string */
DmiBoolean_t     getDescription; /* get optional component
                                   description */
DmiId_t          compId;          /* component ID to start with */
DmiString_t      *className;      /* group class name string
                                   to match*/
DmiAttributeValues_t *keyList;    /* group row keys to match */
```

The *result* parameter is a pointer to a `DmiListComponentsbyClassOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiComponentList_t *reply;      /* list of components */
```

The `DmiListLanguages()` function retrieves the set of language mappings installed for the specified component. The *argIn* parameter is an instance of a `DmiListLanguagesIN` structure containing the following members:

```
DmiHandle_t      handle;          /* An open session handle */
DmiUnsigned_t    maxCount;        /* maximum number to return,
                                   or 0 for all */
DmiId_t          compId;          /* Component to access */
```

The *result* parameter is a pointer to a `DmiListLanguagesOUT` structure containing the following members:

```
DmiErrorStatus_t    error_status;
DmiStringList_t     *reply;          /* List of language strings */
```

The `DmiListClassNames()` function retrieves the class name strings for all groups in a component. This enables the management application to easily determine if a component contains a specific group, or groups. The *argIn* parameter is an instance of a `DmiListClassNamesIN` structure containing the following members:

```
DmiHandle_t         handle;          /* An open session handle */
DmiUnsigned_t       maxCount;        /* maximum number to return,
                                     or 0 for all */
DmiId_t             compId;          /* Component to access */
```

The *result* parameter is a pointer to a `DmiListClassNamesOUT` structure containing the following members:

```
DmiErrorStatus_t    error_status;
DmiClassNameList_t  *reply;          /* List of class names and
                                     group IDs */
```

The `DmiListGroup()` function retrieves a list of groups within a component. With this function you can access a specific group or sequentially access all groups in a component. All enumerations of groups occur within the specified component and do not span components. The *argIn* parameter is an instance of a `DmiListGroupIN` structure containing the following members:

```
DmiHandle_t         handle;          /* An open session handle */
DmiRequestMode_t    requestMode;     /* Unique, first or next group */
DmiUnsigned_t       maxCount;        /* Maximum number to return,
                                     or 0 for all */
DmiBoolean_t        getPragma;       /* Get the optional pragma string */
DmiBoolean_t        getDescription; /* Get optional group description */
DmiId_t             compId;          /* Component to access */
DmiId_t             groupId;         /* Group to start with, refer to
                                     requestMode */
```

The *result* parameter is a pointer to a `DmiListGroupOUT` structure containing the following members:

```
DmiErrorStatus_t    error_status;
DmiGroupList_t      *reply;
```

The caller may choose not to retrieve the group description by setting the value `getDescription` to false. The caller may choose not to retrieve the pragma string by setting the value of `getPragma` to false. The `maxCount`, `requestMode`, and `groupId` parameters allow the caller to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `groupId` specifies the first group requested (or only group if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `groupId` specifies the group just before the one requested. When `requestMode` is `DMI_FIRST`, `groupId` is unused. To control the amount of information returned, the caller sets `maxCount` to something other than zero. The service provider must honor this limit on the amount of information returned. When `maxCount` is zero the service provider returns information for all groups, subject to the constraints imposed by `requestMode` and `groupId`.

DmiListAttributes(3DMI)

The `DmiListAttributes()` function retrieves the properties for one or more attributes in a group. All enumerations of attributes occur within the specified group, and do not span groups. The *argIn* parameter is an instance of a `DmiListAttributesIN` structure containing the following members:

<code>DmiHandle_t</code>	<code>handle;</code>	<code>/* An open session handle */</code>
<code>DmiRequestMode_t</code>	<code>requestMode;</code>	<code>/* Unique, first or next group */</code>
<code>DmiUnsigned_t</code>	<code>maxCount;</code>	<code>/* Maximum number to return,</code> <code>or 0 for all */</code>
<code>DmiBoolean_t</code>	<code>getPragma;</code>	<code>/* Get the optional pragma string */</code>
<code>DmiBoolean_t</code>	<code>getDescription;</code>	<code>/* Get optional group description */</code>
<code>DmiId_t</code>	<code>compId;</code>	<code>/* Component to access */</code>
<code>DmiId_t</code>	<code>groupId;</code>	<code>/* Group to access */</code>
<code>DmiId_t</code>	<code>attribId;</code>	<code>/* Attribute to start with, refer</code> <code>to requestMode */</code>

The *result* parameter is a pointer to a `DmiListAttributesOUT` structure containing the following members:

<code>DmiErrorStatus_t</code>	<code>error_status;</code>	
<code>DmiAttributeList_t</code>	<code>*reply;</code>	<code>/* List of attributes */</code>

You may choose not to retrieve the description string by setting the value of `getDescription` to false. Likewise, you may choose not to retrieve the pragma string by setting the value of `getPragma` to false. The `maxCount`, `requestMode`, and `attribId` parameters allow you to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `attribId` specifies the first attribute requested (or only attribute if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `attribId` specifies the attribute just before the one requested. When `requestMode` is `DMI_FIRST`, `attribId` is unused. To control the amount of information returned, the caller sets `maxCount` to something other than zero. The Service Provider must honor this limit on the amount of information returned. When `maxCount` is zero the service provider returns information for all attributes, subject to the constraints imposed by `requestMode` and `attribId`.

RETURN VALUES

The `DmiListAttributes()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_FILE_ERROR
```

The `DmiListClassNames()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

DmiListAttributes(3DMI)

The `DmiListComponents()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The `DmiListComponentsByClass()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The `DmiListGroups()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_FILE_ERROR
```

The `DmiListLanguages()` function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	Unsafe

SEE ALSO `attributes(5)`

DmiRegisterCi(3DMI)

NAME	DmiRegisterCi, DmiUnRegisterCi, DmiOriginateEvent – Service Provider functions for components
SYNOPSIS	<pre>cc [flag ...] file ... -lci -ldmi -lnsl -lrwtool [library ...] #include <server.h> #include <ciapi.h> extern bool_t DmiRegisterCi(DmiRegisterCiIN <i>argin</i>, DmiRegisterCiOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiUnregisterCi(DmiUnregisterCiIN <i>argin</i>, DmiUnregisterCiOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ; bool_t DmiOriginateEvent(DmiOriginateEventIN <i>argin</i>, DmiOriginateEventOUT *<i>result</i>, DmiRpcHandle *<i>dmi_rpc_handle</i>) ;</pre>
DESCRIPTION	These three functions provide component communication with the DMI through the Component Interface (CI). Component instrumentation code may register with the Service Provider to override its current mechanism for the registered attributes. Instead of manipulating the data in the MIF database or invoking programs, the Service Provider calls the entry points provided in the registration call. Once the component unregisters, the Service Provider returns to a normal method of processing requests for the data as defined in the MIF. Component instrumentation can temporarily interrupt normal processing to perform special functions. Registering attributes through the direct interface overrides attributes that are already being served through the direct interface. RPC is used for communication from the Service Provider to the component instrumentation. For all three functions, <i>argin</i> is the parameter passed to initiate an RPC call, <i>result</i> is the result of the RPC call, and <i>dmi_rpc_handle</i> is an open session RPC handle. The <code>DmiRegisterCi()</code> function registers a callable interface for components that have resident instrumentation code and/or to get the version of the Service Provider. The <code>DmiUnRegisterCi()</code> function communicates to the Service Provider to remove a direct component instrumentation interface from the Service Provider table of registered interfaces. The <code>DmiOriginateEvent()</code> function originates an event for filtering and delivery. Any necessary indication filtering is performed by this function (or by subsequent processing) before the event is forwarded to the management applications. A component ID value of zero (0) specifies the event was generated by something that has not been installed as a component, and has no component ID.
RETURN VALUES	The <code>DmiRegisterCi()</code> function returns the following possible values: <pre>DMIERR_NO_ERROR DMIERR_ILLEGAL_HANDLE DMIERR_OUT_OF_MEMORY</pre>

DmiRegisterCi(3DMI)

DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_DATABASE_CORRUPT
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_DMI_LEVEL

The DmiUnRegisterCi() function returns the following possible values:

DMIERR_NO_ERROR
DMIERR_ILLEGAL_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_UNKNOWN_CI_REGISTRY

The DmiOriginateEvent() function returns the following possible values:

DMIERR_NO_ERROR
DMIERR_ILLEGAL_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_UNKNOWN_CI_REGISTRY

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	Unsafe

SEE ALSO attributes(5)

ea_error(3EXACCT)

NAME	ea_error – error interface to extended accounting library							
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lexacct [<i>library...</i>] #include <exacct.h> int ea_error(void);</pre>							
DESCRIPTION	The <code>ea_error()</code> function returns the error value of the last failure recorded by the invocation of one of the functions of the extended accounting library, <code>libexacct</code> .							
RETURN VALUES	EXR_CORRUPT_FILE	A function failed because the file was not a valid <code>exacct</code> file.						
	EXR_EOF	A function detected the end of the file, either when reading forwards or backwards through the file.						
	EXR_INVALID_BUF	When unpacking an object, an invalid unpack buffer was specified.						
	EXR_INVALID_OBJ	The object type passed to the function is not valid for the requested operation, for example passing a group object to <code>ea_set_item(3EXACCT)</code> .						
	EXR_NO_CREATOR	When creating a new file no creator was specified, or when opening a file for reading the creator value did not match the value in the file.						
	EXR_NOTSUPP	An unsupported type of access was attempted, for example attempting to write to a file that was opened read-only.						
	EXR_OK	The function completed successfully.						
	EXR_SYSCALL_FAIL	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.						
	EXR_UNKN_VERSION	The file referred to by name uses an <code>exacct</code> file version that cannot be processed by this library.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Interface Stability	Evolving							
MT-Level	MT-Safe							
SEE ALSO	<code>read(2)</code> , <code>libexacct(3LIB)</code> , <code>attributes(5)</code>							

NAME	ea_open, ea_close – open or close exacct files								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lexacct [<i>library...</i>] #include <exacct.h> int ea_open(ea_file_t *<i>ef</i>, char *<i>name</i>, char *<i>creator</i>, int <i>aflags</i>, int <i>oflags</i>, mode_t <i>mode</i>); int ea_close(ea_file_t *<i>ef</i>);</pre>								
DESCRIPTION	The <code>ea_open()</code> function provides structured access to exacct files. The <i>aflags</i> argument contains the appropriate exacct flags necessary to describe the file. The <i>oflags</i> and <i>mode</i> arguments contain the appropriate flags and mode to open the file; see <fcntl.h>. If <code>ea_open()</code> is invoked with <code>EO_HEAD</code> specified in <i>aflags</i>, the resulting file is opened with the object cursor located at the first object of the file. If <code>ea_open()</code> is invoked with <code>EO_TAIL</code> specified in <i>aflags</i>, the resulting file is opened with the object cursor positioned beyond the last object in the file. If <code>EO_NO_VALID_HDR</code> is set in <i>aflags</i> along with <code>EO_HEAD</code>, the initial header record will be returned as the first item read from the file. When creating a file, the <i>creator</i> argument should be set (system generated files use the value "SunOS"); when reading a file, this argument should be set to NULL if no validation is required; otherwise it should be set to the expected value in the file. The <code>ea_close()</code> function closes an open exacct file.								
RETURN VALUES	Upon successful completion, <code>ea_open()</code> and <code>ea_close()</code> return 0. Otherwise they return -1 and call <code>ea_error(3EXACCT)</code> to return the extended accounting error value describing the error.								
ERRORS	The <code>ea_open()</code> and <code>ea_close()</code> functions may fail if: <table> <tr> <td><code>EXR_SYSCALL_FAIL</code></td><td>A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.</td></tr> </table> The <code>ea_open()</code> function may fail if: <table> <tr> <td><code>EXR_CORRUPT_FILE</code></td><td>The file referred to by <i>name</i> is not a valid exacct file.</td></tr> <tr> <td><code>EXR_NO_CREATOR</code></td><td>In the case of file creation, the <i>creator</i> argument was NULL. In the case of opening an existing file, a <i>creator</i> argument was not NULL and does not match the <i>creator</i> item of the exacct file.</td></tr> <tr> <td><code>EXR_UNKN_VERSION</code></td><td>The file referred to by <i>name</i> uses an exacct file version that cannot be processed by this library.</td></tr> </table>	<code>EXR_SYSCALL_FAIL</code>	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.	<code>EXR_CORRUPT_FILE</code>	The file referred to by <i>name</i> is not a valid exacct file.	<code>EXR_NO_CREATOR</code>	In the case of file creation, the <i>creator</i> argument was NULL. In the case of opening an existing file, a <i>creator</i> argument was not NULL and does not match the <i>creator</i> item of the exacct file.	<code>EXR_UNKN_VERSION</code>	The file referred to by <i>name</i> uses an exacct file version that cannot be processed by this library.
<code>EXR_SYSCALL_FAIL</code>	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.								
<code>EXR_CORRUPT_FILE</code>	The file referred to by <i>name</i> is not a valid exacct file.								
<code>EXR_NO_CREATOR</code>	In the case of file creation, the <i>creator</i> argument was NULL. In the case of opening an existing file, a <i>creator</i> argument was not NULL and does not match the <i>creator</i> item of the exacct file.								
<code>EXR_UNKN_VERSION</code>	The file referred to by <i>name</i> uses an exacct file version that cannot be processed by this library.								
USAGE	The exacct file format can be used to represent data other than that in the extended accounting format. By using a unique creator type in the file header, application writers can develop their own format suited to the needs of their application.								

ea_open(3EXACCT)

EXAMPLES

EXAMPLE 1 Open and close exacct file.

The following example opens the extended accounting data file for processes. The exacct file is then closed.

```
#include <exacct.h>

ea_file_t ef;
if (ea_open(&ef, "/var/adm/exacct/proc", NULL, EO_HEAD,
           O_RDONLY, 0) == -1)
    exit(1);
(void) ea_close(&ef);
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

ea_error(3EXACCT), ea_pack_object(3EXACCT), ea_set_item(3EXACCT), libexacct(3LIB), attributes(5)

ea_pack_object(3EXACCT)

NAME	ea_pack_object, ea_unpack_object, ea_get_creator, ea_get_hostname, ea_next_object, ea_previous_object, ea_get_object, ea_write_object, ea_copy_object, ea_copy_object_tree, ea_get_object_tree – construct, read, and write extended accounting records
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lexacct [<i>library...</i>] #include <exacct.h> size_t ea_pack_object(ea_object_t *obj, void *buf, size_t bufsize); ea_object_type_t ea_unpack_object(ea_object_t **objp, int flag, void *buf, size_t bufsize); const char *ea_get_creator(ea_file_t *ef); const char *ea_get_hostname(ea_file_t *ef); ea_object_type_t ea_next_object(ea_file_t *ef, ea_object_t *obj); ea_object_type_t ea_previous_object(ea_file_t *ef, ea_object_t *obj); ea_object_type_t ea_get_object(ea_file_t *ef, ea_object_t *obj); int ea_write_object(ea_file_t *ef, ea_object_t *obj); ea_object_type_t *ea_copy_object(const ea_object_t *src); ea_object_type_t *ea_copy_object_tree(const ea_object_t *src); ea_object_type_t *ea_get_object_tree(ea_file_t *ef, uint32_t nobj);</pre>
DESCRIPTION	The <code>ea_pack_object()</code> function converts <code>exacct</code> objects from their in-memory representation to their file representation. It is passed an object pointer that points to the top of an <code>exacct</code> object hierarchy representing one or more <code>exacct</code> records. It returns the size of the buffer required to contain the packed buffer representing the object hierarchy. To obtain the correct size of the required buffer, the <code>buf</code> and <code>bufsize</code> parameters can be set to <code>NULL</code> and <code>0</code> respectively, and the required buffer size will be returned. The resulting packed record can be passed to <code>putacct(2)</code> or to <code>ea_set_item(3EXACCT)</code> when constructing an object of type <code>EXT_EXACCT_OBJECT</code>. The <code>ea_unpack_object()</code> function reverses the packing process performed by <code>ea_pack_object()</code>. A packed buffer passed to <code>ea_unpack_object()</code> is unpacked into the original hierarchy of objects. If the unpack operation fails (for example, due to a corrupted or incomplete buffer), it returns <code>EO_ERROR</code>; otherwise, the object type of the first object in the hierarchy is returned. If <code>ea_unpack_object()</code> is invoked with <code>flag</code> equal to <code>EUP_ALLOC</code>, it allocates memory for the variable-length data in the included objects. Otherwise, with <code>flag</code> equal to <code>EUP_NOALLOC</code>, it sets the variable length data pointers within the unpacked object structures to point within the buffer indicated by <code>buf</code>. In both cases, <code>ea_unpack_object()</code> allocates all the necessary <code>exacct</code> objects to represent the unpacked record. The resulting object hierarchy can be freed using <code>ea_free_object(3EXACCT)</code> with the same <code>flag</code> value.

ea_pack_object(3EXACCT)

The `ea_get_creator()` function returns a pointer to a string representing the recorded creator of the `exacct` file. The `ea_get_hostname()` function returns a pointer to a string representing the recorded hostname on which the `exacct` file was created. These functions will return `NULL` if their respective field was not recorded in the `exacct` file header.

The `ea_next_object()` function reads the basic fields (`eo_catalog` and `eo_type`) into the `ea_object_t` indicated by *obj* from the `exacct` file referred to by *ef* and rewinds to the head of the record. If the read object is corrupted, `ea_next_object()` returns `EO_ERROR` and records the extended accounting error code, accessible with `ea_error(3EXACCT)`. If end-of-file is reached, `EO_ERROR` is returned and the extended accounting error code is set to `EXR_EOF`.

The `ea_previous_object()` function skips back one object in the file and reads its basic fields (`eo_catalog` and `eo_type`) into the indicated `ea_object_t`. If the read object is corrupted, `ea_previous_object()` returns `EO_ERROR` and records the extended accounting error code, accessible with `ea_error(3EXACCT)`. If end-of-file is reached, `EO_ERROR` is returned and the extended accounting error code is set to `EXR_EOF`.

The `ea_get_object()` function reads the value fields into the `ea_object_t` indicated by *obj*, allocating memory as necessary, and advances to the head of the next record. Once a record group object is retrieved using `ea_get_object()`, subsequent calls to `ea_get_object()` and `ea_next_object()` will track through the objects within the record group, and on reaching the end of the group, will return the next object at the same level as the group from the file. If the read object is corrupted, `ea_get_object()` returns `EO_ERROR` and records the extended accounting error code, accessible with `ea_error(3EXACCT)`. If end-of-file is reached, `EO_ERROR` is returned and the extended accounting error code is set to `EXR_EOF`.

The `ea_write_object()` function appends the given object to the open `exacct` file indicated by *ef* and returns 0. If the write fails, `ea_write_object()` returns -1 and sets the extended accounting error code to indicate the error, accessible with `ea_error(3EXACCT)`.

The `ea_copy_object()` function copies an `ea_object_t`. If the source object is part of a chain, only the current object is copied. If the source object is a group, only the group object is copied without its list of members and the `eg_nobjs` and `eg_objs` fields are set to 0 and `NULL`, respectively. Use `ea_copy_tree()` to copy recursively a group or a list of items.

The `ea_copy_object_tree()` function recursively copies an `ea_object_t`. All elements in the `eo_next` list are copied, and any group objects are recursively copied. The returned object can be completely freed with `ea_free_object(3EXACCT)` by specifying the `EUP_ALLOC` flag.

The `ea_get_object_tree()` function reads in *nobj* top-level objects from the file, returning the same data structure that would have originally been passed to `ea_write_object()`. On encountering a group object, the `ea_get_object()`

ea_pack_object(3EXACCT)

function reads only the group header part of the group, whereas `ea_get_object_tree()` reads the group and all its member items, recursing into sub-records if necessary. The returned object data structure can be completely freed with `ea_free_object()` by specifying the `EUP_ALLOC` flag.

RETURN VALUES

The `ea_pack_object()` function returns the number of bytes required to hold the `exacct` object being operated upon. If the returned size exceeds *bufsize*, the pack operation does not complete and the function returns `(size_t)-1` and sets the extended accounting error code to indicate the error.

The `ea_get_object()` function returns the `ea_object_type` of the object if the object was retrieved successfully. Otherwise, it returns `EO_ERROR` and sets the extended accounting error code to indicate the error.

The `ea_next_object()` function returns the `ea_object_type` of the next `exacct` object in the file. It returns `EO_ERROR` if the `exacct` file is corrupted sets the extended accounting error code to indicate the error.

The `ea_unpack_object()` function returns the `ea_object_type` of the first `exacct` object unpacked from the buffer. It returns `EO_ERROR` if the `exacct` file is corrupted, and sets the extended accounting error code to indicate the error.

The `ea_write_object()` function returns 0 on success. Otherwise it returns -1 and sets the extended accounting error code to indicate the error.

The `ea_copy_object()` and `ea_copy_object_tree()` functions return the copied object on success. Otherwise they return `NULL` and set the extended accounting error code to indicate the error.

The `ea_get_object_tree()` function returns the list of objects read from the file on success. Otherwise it returns `NULL` and sets the extended accounting error code to indicate the error.

The extended account error code can be retrieved using `ea_error(3EXACCT)`.

ERRORS

These functions may fail if:

<code>EXR_SYSCALL_FAIL</code>	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.
<code>EXR_CORRUPT_FILE</code>	The file referred to by <i>name</i> is not a valid <code>exacct</code> file, or is unparsable, and therefore appears corrupted. This error is also used by <code>ea_unpack_buffer()</code> to indicate a corrupted buffer.
<code>EXR_NO_MEMORY</code>	A memory allocation required to complete the operation failed.
<code>EXR_EOF</code>	The end of the file has been reached. In the case of <code>ea_previous_record()</code> , the previous record could

ea_pack_object(3EXACCT)

not be reached, either because the head of the file was encountered or because the previous record could not be skipped over.

USAGE The `exacct` file format can be used to represent data other than that in the extended accounting format. By using a unique creator type in the file header, application writers can develop their own format suited to the needs of their application.

EXAMPLES **EXAMPLE 1** Open and close `exacct` file.

The following example opens the extended accounting data file for processes. The `exacct` file is then closed.

```
#include <stdio.h>
#include <exacct.h>

ea_file_t ef;
ea_object_t *obj;

...

ea_open(&ef, "foo", O_RDONLY, ...);

while ((obj = ea_get_object_tree(&ef, 1)) != NULL) {
    if (obj->eo_type == EO_ITEM) {
        /* handle item */
    } else {
        /* handle group */
    }
    ea_free_object(obj, EUP_ALLOC);
}

if (ea_error() != EXR_EOF) {
    /* handle error */
}

ea_close(&ef);
```

EXAMPLE 2 Construct an `exacct` file consisting of a single object containing the current process ID.

```
#include <sys/types.h>
#include <unistd.h>
#include <exacct.h>

...

ea_file_t ef;
ea_object_t obj;
pid_t my_pid;

ea_open(&ef, "foo", O_CREAT | O_WRONLY, ...);

my_pid = getpid();
ea_set_item(&obj, EXT_UINT32 | EXC_DEFAULT | EXT_PROC_PID, &my_pid, 0);
```


ea_pack_object(3EXACCT)

EXAMPLE 2 Construct an exacct file consisting of a single object containing the current process ID. *(Continued)*

```
(void) ea_write_object(&ef, &obj);  
ea_close(&ef);  
...
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO read(2), ea_error(3EXACCT), ea_open(3EXACCT), ea_set_item(3EXACCT), libexacct(3LIB), attributes(5)

ea_set_item(3EXACCT)

NAME	ea_set_item, ea_alloc, ea_strdup, ea_set_group, ea_match_object_catalog, ea_attach_to_object, ea_attach_to_group, ea_free, ea_strfree, ea_free_item, ea_free_object – create, destroy and manipulate exacct objects
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lexacct [<i>library...</i>] #include <exacct.h> int ea_set_item(ea_object_t *obj, ea_catalog_t tag, void *value, size_t valsize); void *ea_alloc(size_t size); char *ea_strdup(char *ptr); int ea_set_group(ea_object_t *obj, ea_catalog_t tag); int ea_match_object_catalog(ea_object_t *obj, ea_catalog_t catmask); void ea_attach_to_object(ea_object_t *head_obj, ea_object_t *obj); void ea_attach_to_group(ea_object_t *group_obj, ea_object_t *obj); void ea_free(void *ptr, size_t size); void ea_strfree(char *ptr); int ea_free_item(ea_object_t *obj, int flag); void ea_free_object(ea_object_t *obj, int flag);</pre>
DESCRIPTION	The <code>ea_alloc()</code> function allocates a block of memory of the requested size. This block can be safely passed to <code>libexacct</code> functions, and can be safely freed by any of the <code>ea_free()</code> functions. The <code>ea_strdup()</code> function can be used to duplicate a string that is to be stored inside an <code>ea_object_t</code> structure. The <code>ea_set_item()</code> function assigns the given <code>exacct</code> object to be a data item with <i>value</i> set according to the remaining arguments. For buffer-based data values (<code>EXT_STRING</code>, <code>EXT_EXACCT_OBJECT</code>, and <code>EXT_RAW</code>), a copy of the passed buffer is taken. In the case of <code>EXT_EXACCT_OBJECT</code>, the passed buffer should be a packed <code>exacct</code> object as returned by <code>ea_pack_object(3EXACCT)</code>. Any item assigned with <code>ea_set_item()</code> should be freed with <code>ea_free_item()</code> specifying a flag value of <code>EUP_ALLOC</code> when the item is no longer needed. The <code>ea_match_object_catalog()</code> function returns <code>TRUE</code> if the <code>exacct</code> object specified by <i>obj</i> has a catalog tag that matches the mask specified by <i>catmask</i>. The <code>ea_attach_to_object()</code> function attaches an object to the given object. The <code>ea_attach_to_group()</code> function attaches a chain of objects as member items of the given group. Objects are inserted at the end of the list of any previously attached objects.

ea_set_item(3EXACCT)

	The <code>ea_free()</code> function frees a block of memory previously allocated by <code>ea_alloc()</code>. The <code>ea_strfree()</code> function frees a string previously copied by <code>ea_strdup()</code>. The <code>ea_free_item()</code> function frees the <i>value</i> fields in the <code>ea_object_t</code> indicated by <i>obj</i>, if <code>EUP_ALLOC</code> is specified. The object itself is not freed. The <code>ea_free_object()</code> function frees the specified object and any attached hierarchy of objects. If the <i>flag</i> argument is set to <code>EUP_ALLOC</code>, <code>ea_free_object()</code> will also free any variable-length data in the object hierarchy; if set to <code>EUP_NOALLOC</code>, <code>ea_free_object()</code> will not free variable-length data. In particular, these flags should correspond to those specified in calls to <code>ea_unpack_object(3EXACCT)</code>.				
RETURN VALUES	The <code>ea_match_object_catalog()</code> function returns 0 if the object's catalog tag does not match the given mask, and 1 if there is a match. Other integer-valued functions return 0 if successful. Otherwise these functions return -1 and set the extended accounting error code appropriately. Pointer-valued functions return a valid pointer if successful and NULL otherwise, setting the extended accounting error code appropriately. The extended accounting error code can be examined with <code>ea_error(3EXACCT)</code>.				
ERRORS	The <code>ea_set_item()</code>, <code>ea_set_group()</code>, and <code>ea_match_object_catalog()</code> functions may fail if: <table><tr><td><code>EXR_SYSCALL_FAIL</code></td><td>A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.</td></tr><tr><td><code>EXR_INVALID_OBJECT</code></td><td>The passed object is of an incorrect type, for example passing a group object to <code>ea_set_item()</code>.</td></tr></table>	<code>EXR_SYSCALL_FAIL</code>	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.	<code>EXR_INVALID_OBJECT</code>	The passed object is of an incorrect type, for example passing a group object to <code>ea_set_item()</code> .
<code>EXR_SYSCALL_FAIL</code>	A system call invoked by the function failed. The <code>errno</code> variable contains the error value set by the underlying call.				
<code>EXR_INVALID_OBJECT</code>	The passed object is of an incorrect type, for example passing a group object to <code>ea_set_item()</code> .				
USAGE	The <code>exacct</code> file format can be used to represent data other than that in the extended accounting format. By using a unique creator type in the file header, application writers can develop their own format suited to the needs of their application.				
EXAMPLES	EXAMPLE 1 Open and close <code>exacct</code> file. Construct an <code>exacct</code> file consisting of a single object containing the current process ID. <pre>#include <sys/types.h> #include <unistd.h> #include <exacct.h> ... ea_file_t ef; ea_object_t obj; pid_t my_pid; my_pid = getpid(); ea_set_item(&obj, EXT_UINT32 EXC_DEFAULT EXT_PROC_PID, &my_pid, sizeof(my_pid));</pre>				

ea_set_item(3EXACCT)

EXAMPLE 1 Open and close exacct file. *(Continued)*

...

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO read(2), ea_error(3EXACCT), ea_open(3EXACCT), ea_pack_object(3EXACCT), libexacct(3LIB), attributes(5)

NAME	efi_alloc_and_init, efi_alloc_and_read, efi_free, efi_write – manipulate a disk's EFI Partition Table				
SYNOPSIS	<pre>cc [flag ...] file... -lefi [library ...] #include <sys/vtoc.h> #include <sys/efi_partition.h> int efi_alloc_and_init(int fd, uint32_t nparts, dk_gpt_t **vtoc); int efi_alloc_and_read(int fd, dk_gpt_t **vtoc); void efi_free(dk_gpt_t *vtoc); int efi_write(int fd, dk_gpt_t *vtoc);</pre>				
DESCRIPTION	The <code>efi_alloc_and_init()</code> function initializes the <code>dk_gpt_t</code> structure specified by <code>vtoc</code> in preparation for a call to <code>efi_write()</code>. It calculates and initializes the <code>efi_version</code>, <code>efi_lbasize</code>, <code>efi_nparts</code>, <code>efi_first_u_lba</code>, <code>efi_last_lba</code>, and <code>efi_last_u_lba</code> members of this structure. The caller can then set the <code>efi_nparts</code> member. The <code>efi_alloc_and_read()</code> function allocates memory and returns the partition table. The <code>efi_free()</code> function frees the memory allocated by <code>efi_alloc_and_init()</code> and <code>efi_alloc_and_read()</code>. The <code>efi_write()</code> function writes the EFI partition table. The <code>fd</code> argument refers to any slice on a raw disk, opened with <code>O_NDELAY</code>. See <code>open(2)</code>. The <code>nparts</code> argument specifies the number of desired partitions. The <code>vtoc</code> argument is a <code>dk_gpt_t</code> structure that describes an EFI partition table and contains at least the following members: <pre>uint_t efi_version; /* set to EFI_VERSION_CURRENT */ uint_t efi_nparts; /* number of partitions in efi_parts */ uint_t efi_lbasize; /* size of block in bytes */ diskaddr_t efi_last_lba; /* last block on the disk */ diskaddr_t efi_first_u_lba; /* first block after labels */ diskaddr_t efi_last_u_lba; /* last block before backup labels */ struct dk_part efi_parts[]; /* array of partitions */</pre> RETURN VALUES Upon successful completion, <code>efi_alloc_and_init()</code> returns 0. Otherwise it returns <code>VT_EIO</code> if an I/O operation to the disk fails. Upon successful completion, <code>efi_alloc_and_read()</code> returns a positive integer indicating the slice index associated with the open file descriptor. Otherwise, it returns a negative integer to indicate one of the following: <table> <tr> <td><code>VT_EIO</code></td><td>An I/O error occurred.</td></tr> <tr> <td><code>VT_ERROR</code></td><td>An unknown error occurred.</td></tr> </table>	<code>VT_EIO</code>	An I/O error occurred.	<code>VT_ERROR</code>	An unknown error occurred.
<code>VT_EIO</code>	An I/O error occurred.				
<code>VT_ERROR</code>	An unknown error occurred.				

efi_alloc_and_init(3EXT)

VT_EINVAL An EFI label was not found.

Upon successful completion, `efi_write()` returns 0. Otherwise, it returns a negative integer to indicate one of the following:

VT_EIO An I/O error occurred.

VT_ERROR An unknown error occurred.

VT_EINVAL The label contains incorrect data.

USAGE The EFI label is used on disks with more than $1^{32}-1$ blocks. For compatibility reasons, the `read_vtoc(3EXT)` and `write_vtoc()` functions should be used on smaller disks. The application should attempt the `read_vtoc()` or `write_vtoc()` call, check for an error of `VT_ENOTSUP`, then call the analogous EFI function.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	Unsafe

SEE ALSO `fmthard(1M)`, `format(1M)`, `prtvtoc(1M)`, `ioctl(2)`, `open(2)`, `libefi(3LIB)`, `read_vtoc(3EXT)`, `attributes(5)`, `dkio(7I)`

elf32_checksum(3ELF)

NAME	elf32_checksum, elf64_checksum – return checksum of elf image						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> long elf32_checksum(Elf *elf) ; long elf64_checksum(Elf *elf) ;</pre>						
DESCRIPTION	The <code>elf32_checksum()</code> function returns a simple checksum of selected sections of the image identified by <i>elf</i>. The value is typically used as the <code>.dynamic</code> tag <code>DT_CHECKSUM</code>, recorded in dynamic executables and shared objects. Selected sections of the image are used to calculate the checksum in order that its value is not affected by utilities such as <code>strip(1)</code>. For the 64-bit class, replace 32 with 64 as appropriate.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>elf_version(3ELF)</code> , <code>gelf(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						

elf32_fsize(3ELF)

NAME	elf32_fsize, elf64_fsize – return the size of an object file type																					
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> size_t elf32_fsize(Elf_Type <i>type</i>, size_t <i>count</i>, unsigned <i>ver</i>); size_t elf64_fsize(Elf_Type <i>type</i>, size_t <i>count</i>, unsigned <i>ver</i>);</pre>																					
DESCRIPTION	elf32_fsize() gives the size in bytes of the 32-bit file representation of <i>count</i> data objects with the given type. The library uses version <i>ver</i> to calculate the size. See elf(3ELF) and elf_version(3ELF). Constant values are available for the sizes of fundamental types: <table><tr><td>Elf_Type</td><td>File Size</td><td>Memory Size</td></tr><tr><td>ELF_T_ADDR</td><td>ELF32_FSZ_ADDR</td><td>sizeof(Elf32_Addr)</td></tr><tr><td>ELF_T_BYTE</td><td>1</td><td>sizeof(unsigned char)</td></tr><tr><td>ELF_T_HALF</td><td>ELF32_FSZ_HALF</td><td>sizeof(Elf32_Half)</td></tr><tr><td>ELT_T_OFF</td><td>ELF32_FSZ_OFF</td><td>sizeof(Elf32_Off)</td></tr><tr><td>ELF_T_SWORD</td><td>ELF32_FSZ_SWORD</td><td>sizeof(Elf32_Sword)</td></tr><tr><td>ELF_T_WORD</td><td>ELF32_FSZ_WORD</td><td>sizeof(Elf32_Word)</td></tr></table> elf32_fsize() returns 0 if the value of <i>type</i> or <i>ver</i> is unknown. See elf32_xlatetof(3ELF) for a list of the type values. For the 64-bit class, replace 32 with 64 as appropriate.	Elf_Type	File Size	Memory Size	ELF_T_ADDR	ELF32_FSZ_ADDR	sizeof(Elf32_Addr)	ELF_T_BYTE	1	sizeof(unsigned char)	ELF_T_HALF	ELF32_FSZ_HALF	sizeof(Elf32_Half)	ELT_T_OFF	ELF32_FSZ_OFF	sizeof(Elf32_Off)	ELF_T_SWORD	ELF32_FSZ_SWORD	sizeof(Elf32_Sword)	ELF_T_WORD	ELF32_FSZ_WORD	sizeof(Elf32_Word)
Elf_Type	File Size	Memory Size																				
ELF_T_ADDR	ELF32_FSZ_ADDR	sizeof(Elf32_Addr)																				
ELF_T_BYTE	1	sizeof(unsigned char)																				
ELF_T_HALF	ELF32_FSZ_HALF	sizeof(Elf32_Half)																				
ELT_T_OFF	ELF32_FSZ_OFF	sizeof(Elf32_Off)																				
ELF_T_SWORD	ELF32_FSZ_SWORD	sizeof(Elf32_Sword)																				
ELF_T_WORD	ELF32_FSZ_WORD	sizeof(Elf32_Word)																				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe															
ATTRIBUTE TYPE	ATTRIBUTE VALUE																					
Interface Stability	Stable																					
MT-Level	MT-Safe																					
SEE ALSO	elf(3ELF), elf32_xlatetof(3ELF), elf_version(3ELF), libelf(3LIB), attributes(5)																					

NAME	elf32_getehdr, elf32_newehdr, elf64_getehdr, elf64_newehdr – retrieve class-dependent object file header						
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf32_Ehdr *elf32_getehdr(Elf *elf); Elf32_Ehdr *elf32_newehdr(Elf *elf); Elf64_Ehdr *elf64_getehdr(Elf *elf); Elf64_Ehdr *elf64_newehdr(Elf *elf);</pre>						
DESCRIPTION	For a 32-bit class file, <code>elf32_getehdr()</code> returns a pointer to an ELF header, if one is available for the ELF descriptor <code>elf</code>. If no header exists for the descriptor, <code>elf32_newehdr()</code> allocates a clean one, but it otherwise behaves the same as <code>elf32_getehdr()</code>. It does not allocate a new header if one exists already. If no header exists for <code>elf32_getehdr()</code>, one cannot be created for <code>elf32_newehdr()</code>, a system error occurs, the file is not a 32-bit class file, or <code>elf</code> is null, both functions return a null pointer. For the 64-bit class, replace 32 with 64 as appropriate. The header includes the following members: <pre>unsigned char e_ident[EI_NIDENT]; Elf32_Half e_type; Elf32_Half e_machine; Elf32_Word e_version; Elf32_Addr e_entry; Elf32_Off e_phoff; Elf32_Off e_shoff; Elf32_Word e_flags; Elf32_Word e_ehsize; Elf32_Half e_phentsize; Elf32_Half e_phnum; Elf32_Half e_shentsize; Elf32_Half e_shnum; Elf32_Half e_shstrndx;</pre> <code>elf32_newehdr()</code> automatically sets the <code>ELF_F_DIRTY</code> bit. See <code>elf_flagdata(3ELF)</code>. A program may use <code>elf_getident()</code> to inspect the identification bytes from a file.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						

elf32_getehdr(3ELF)

SEE ALSO elf(3ELF), elf_begin(3ELF), elf_flagdata(3ELF), elf_getident(3ELF),
libelf(3LIB), attributes(5)

NAME	elf32_getphdr, elf32_newphdr, elf64_getphdr, elf64_newphdr – retrieve class-dependent program header table
SYNOPSIS	<pre>cc [flag ...] file... -lelf [library ...] #include <libelf.h> Elf32_Phdr *elf32_getphdr(Elf *elf); Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count); Elf64_Phdr *elf64_getphdr(Elf *elf); Elf64_Phdr *elf64_newphdr(Elf *elf, size_t count);</pre>
DESCRIPTION	For a 32-bit class file, <code>elf32_getphdr()</code> returns a pointer to the program execution header table, if one is available for the ELF descriptor <i>elf</i>. <code>elf32_newphdr()</code> allocates a new table with <i>count</i> entries, regardless of whether one existed previously, and sets the <code>ELF_F_DIRTY</code> bit for the table. See <code>elf_flagdata(3ELF)</code>. Specifying a zero <i>count</i> deletes an existing table. Note this behavior differs from that of <code>elf32_newehdr()</code> allowing a program to replace or delete the program header table, changing its size if necessary. See <code>elf32_getehdr(3ELF)</code>. If no program header table exists, the file is not a 32-bit class file, an error occurs, or <i>elf</i> is <code>NULL</code>, both functions return a null pointer. Additionally, <code>elf32_newphdr()</code> returns a null pointer if <i>count</i> is 0. The table is an array of <code>Elf32_Phdr</code> structures, each of which includes the following members: <pre>Elf32_Word p_type; Elf32_Off p_offset; Elf32_Addr p_vaddr; Elf32_Addr p_paddr; Elf32_Word p_filesz; Elf32_Word p_memsz; Elf32_Word p_flags; Elf32_Word p_align;</pre> The <code>Elf64_Phdr</code> structures include the following members: <pre>Elf64_Word p_type; Elf64_Word p_flags; Elf64_Off p_offset; Elf64_Addr p_vaddr; Elf64_Addr p_paddr; Elf64_Xword p_filesz; Elf64_Xword p_memsz; Elf64_Xword p_align;</pre> For the 64-bit class, replace 32 with 64 as appropriate.

elf32_getphdr(3ELF)

The ELF header's `e_phnum` member tells how many entries the program header table has. See `elf32_getehdr(3ELF)`. A program may inspect this value to determine the size of an existing table; `elf32_newphdr()` automatically sets the member's value to *count*. If the program is building a new file, it is responsible for creating the file's ELF header before creating the program header table.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO `elf(3ELF)`, `elf32_getehdr(3ELF)`, `elf_begin(3ELF)`, `elf_flagdata(3ELF)`, `libelf(3LIB)`, `attributes(5)`

NAME	elf32_getshdr, elf64_getshdr – retrieve class-dependent section header						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> Elf32_Shdr *elf32_getshdr(Elf_Scn *scn); Elf64_Shdr *elf64_getshdr(Elf_Scn *scn);</pre>						
DESCRIPTION	For a 32-bit class file, <code>elf32_getshdr()</code> returns a pointer to a section header for the section descriptor <code>scn</code>. Otherwise, the file is not a 32-bit class file, <code>scn</code> was NULL, or an error occurred; <code>elf32_getshdr()</code> then returns NULL. The <code>elf32_getshdr</code> header includes the following members: <pre>Elf32_Word sh_name; Elf32_Word sh_type; Elf32_Word sh_flags; Elf32_Addr sh_addr; Elf32_Off sh_offset; Elf32_Word sh_size; Elf32_Word sh_link; Elf32_Word sh_info; Elf32_Word sh_addralign; Elf32_Word sh_entsize;</pre> while the <code>elf64_getshdr</code> header includes the following members: <pre>Elf64_Word sh_name; Elf64_Word sh_type; Elf64_Xword sh_flags; Elf64_Addr sh_addr; Elf64_Off sh_offset; Elf64_Xword sh_size; Elf64_Word sh_link; Elf64_Word sh_info; Elf64_Xword sh_addralign; Elf64_Xword sh_entsize;</pre> For the 64-bit class, replace 32 with 64 as appropriate. If the program is building a new file, it is responsible for creating the file's ELF header before creating sections.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						

elf32_getshdr(3ELF)

SEE ALSO elf(3ELF), elf_flagdata(3ELF), elf_getscn(3ELF), elf_strptr(3ELF),
libelf(3LIB), attributes(5)

NAME	elf32_xlatetof, elf32_xlatetom, elf64_xlatetof, elf64_xlatetom – class-dependent data translation								
SYNOPSIS	<pre>cc [flag ...] file... -lelf [library ...] #include <libelf.h> Elf_Data *elf32_xlatetof(Elf_Data *dst, const Elf_Data *src, unsigned encode); Elf_Data *elf32_xlatetom(Elf_Data *dst, const Elf_Data *src, unsigned encode); Elf_Data *elf64_xlatetof(Elf_Data *dst, const Elf_Data *src, unsigned encode); Elf_Data *elf64_xlatetom(Elf_Data *dst, const Elf_Data *src, unsigned encode);</pre>								
DESCRIPTION	elf32_xlatetom() translates various data structures from their 32-bit class file representations to their memory representations; elf32_xlatetof() provides the inverse. This conversion is particularly important for cross development environments. <i>src</i> is a pointer to the source buffer that holds the original data; <i>dst</i> is a pointer to a destination buffer that will hold the translated copy. <i>encode</i> gives the byte encoding in which the file objects are to be represented and must have one of the encoding values defined for the ELF header's <i>e_ident</i> [EI_DATA] entry (see elf_getident(3ELF)). If the data can be translated, the functions return <i>dst</i>. Otherwise, they return NULL because an error occurred, such as incompatible types, destination buffer overflow, etc. elf_getdata(3ELF) describes the Elf_Data descriptor, which the translation routines use as follows: <table> <tr> <td>d_buf</td><td>Both the source and destination must have valid buffer pointers.</td></tr> <tr> <td>d_type</td><td>This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.</td></tr> <tr> <td>d_size</td><td>This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The translation routines reset the destination's d_size member to the actual size required, after the translation occurs. The source and destination sizes may differ.</td></tr> <tr> <td>d_version</td><td>This member holds the version number of the objects (desired) in the buffer. The source and destination versions are independent.</td></tr> </table> Translation routines allow the source and destination buffers to coincide. That is, <i>dst</i>→<i>d_buf</i> may equal <i>src</i>→<i>d_buf</i>. Other cases where the source and destination buffers overlap give undefined behavior.	d_buf	Both the source and destination must have valid buffer pointers.	d_type	This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.	d_size	This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The translation routines reset the destination's d_size member to the actual size required, after the translation occurs. The source and destination sizes may differ.	d_version	This member holds the version number of the objects (desired) in the buffer. The source and destination versions are independent.
d_buf	Both the source and destination must have valid buffer pointers.								
d_type	This member's value specifies the type of the data to which d_buf points and the type of data to be created in the destination. The program supplies a d_type value in the source; the library sets the destination's d_type to the same value. These values are summarized below.								
d_size	This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The translation routines reset the destination's d_size member to the actual size required, after the translation occurs. The source and destination sizes may differ.								
d_version	This member holds the version number of the objects (desired) in the buffer. The source and destination versions are independent.								

elf32_xlatetof(3ELF)

Elf_Type	32-Bit Memory Type
ELF_T_ADDR	Elf32_Addr
ELF_T_BYTE	unsigned char
ELF_T_DYN	Elf32_Dyn
ELF_T_EHDR	Elf32_Ehdr
ELF_T_HALF	Elf32_Half
ELF_T_OFF	Elf32_Off
ELF_T_PHDR	Elf32_Phdr
ELF_T_REL	Elf32_Rel
ELF_T_RELA	Elf32_Rela
ELF_T_SHDR	Elf32_Shdr
ELF_T_SWORD	Elf32_Sword
ELF_T_SYM	Elf32_Sym
ELF_T_WORD	Elf32_Word

Translating buffers of type ELF_T_BYTE does not change the byte order.

For the 64-bit class, replace 32 with 64 as appropriate.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO

elf(3ELF), elf32_fsize(3ELF), elf_getdata(3ELF), elf_getident(3ELF), libelf(3LIB), attributes(5)

NAME	elf – object file access library														
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h></pre>														
DESCRIPTION	Functions in the ELF access library let a program manipulate ELF (Executable and Linking Format) object files, archive files, and archive members. The header provides type and function declarations for all library services. Programs communicate with many of the higher-level routines using an <i>ELF descriptor</i>. That is, when the program starts working with a file, <code>elf_begin(3ELF)</code> creates an ELF descriptor through which the program manipulates the structures and information in the file. These ELF descriptors can be used both to read and to write files. After the program establishes an ELF descriptor for a file, it may then obtain <i>section descriptors</i> to manipulate the sections of the file (see <code>elf_getscn(3ELF)</code>). Sections hold the bulk of an object file’s real information, such as text, data, the symbol table, and so on. A section descriptor “belongs” to a particular ELF descriptor, just as a section belongs to a file. Finally, <i>data descriptors</i> are available through section descriptors, allowing the program to manipulate the information associated with a section. A data descriptor “belongs” to a section descriptor. Descriptors provide private handles to a file and its pieces. In other words, a data descriptor is associated with one section descriptor, which is associated with one ELF descriptor, which is associated with one file. Although descriptors are private, they give access to data that may be shared. Consider programs that combine input files, using incoming data to create or update another file. Such a program might get data descriptors for an input and an output section. It then could update the output descriptor to reuse the input descriptor’s data. That is, the descriptors are distinct, but they could share the associated data bytes. This sharing avoids the space overhead for duplicate buffers and the performance overhead for copying data unnecessarily.														
File Classes	ELF provides a framework in which to define a family of object files, supporting multiple processors and architectures. An important distinction among object files is the <i>class</i>, or capacity, of the file. The 32-bit class supports architectures in which a 32-bit object can represent addresses, file sizes, and so on, as in the following: <table><tr><th>Name</th><th>Purpose</th></tr><tr><td>Elf32_Addr</td><td>Unsigned address</td></tr><tr><td>Elf32_Half</td><td>Unsigned medium integer</td></tr><tr><td>Elf32_Off</td><td>Unsigned file offset</td></tr><tr><td>Elf32_Sword</td><td>Signed large integer</td></tr><tr><td>Elf32_Word</td><td>Unsigned large integer</td></tr><tr><td>unsigned char</td><td>Unsigned small integer</td></tr></table>	Name	Purpose	Elf32_Addr	Unsigned address	Elf32_Half	Unsigned medium integer	Elf32_Off	Unsigned file offset	Elf32_Sword	Signed large integer	Elf32_Word	Unsigned large integer	unsigned char	Unsigned small integer
Name	Purpose														
Elf32_Addr	Unsigned address														
Elf32_Half	Unsigned medium integer														
Elf32_Off	Unsigned file offset														
Elf32_Sword	Signed large integer														
Elf32_Word	Unsigned large integer														
unsigned char	Unsigned small integer														

elf(3ELF)

**Data
Representation**

The 64-bit class works the same as the 32-bit class, substituting 64 for 32 as necessary. Other classes will be defined as necessary, to support larger (or smaller) machines. Some library services deal only with data objects for a specific class, while others are class-independent. To make this distinction clear, library function names reflect their status, as described below.

Conceptually, two parallel sets of objects support cross compilation environments. One set corresponds to file contents, while the other set corresponds to the native memory image of the program manipulating the file. Type definitions supplied by the headers work on the native machine, which may have different data encodings (size, byte order, and so on) than the target machine. Although native memory objects should be at least as big as the file objects (to avoid information loss), they may be bigger if that is more natural for the host machine.

Translation facilities exist to convert between file and memory representations. Some library routines convert data automatically, while others leave conversion as the program's responsibility. Either way, programs that create object files must write file-typed objects to those files; programs that read object files must take a similar view. See `elf32_xlatetof(3ELF)` and `elf32_fsize(3ELF)` for more information.

Programs may translate data explicitly, taking full control over the object file layout and semantics. If the program prefers not to have and exercise complete control, the library provides a higher-level interface that hides many object file details. `elf_begin()` and related functions let a program deal with the native memory types, converting between memory objects and their file equivalents automatically when reading or writing an object file.

ELF Versions

Object file versions allow ELF to adapt to new requirements. *Three independent versions* can be important to a program. First, an application program knows about a particular version by virtue of being compiled with certain headers. Second, the access library similarly is compiled with header files that control what versions it understands. Third, an ELF object file holds a value identifying its version, determined by the ELF version known by the file's creator. Ideally, all three versions would be the same, but they may differ.

If a program's version is newer than the access library, the program might use information unknown to the library. Translation routines might not work properly, leading to undefined behavior. This condition merits installing a new library.

The library's version might be newer than the program's and the file's. The library understands old versions, thus avoiding compatibility problems in this case.

Finally, a file's version might be newer than either the program or the library understands. The program might or might not be able to process the file properly, depending on whether the file has extra information and whether that information can be safely ignored. Again, the safe alternative is to install a new library that understands the file's version.

	To accommodate these differences, a program must use <code>elf_version(3ELF)</code> to pass its version to the library, thus establishing the <i>working version</i> for the process. Using this, the library accepts data from and presents data to the program in the proper representations. When the library reads object files, it uses each file's version to interpret the data. When writing files or converting memory types to the file equivalents, the library uses the program's working version for the file data.	
System Services	As mentioned above, <code>elf_begin()</code> and related routines provide a higher-level interface to ELF files, performing input and output on behalf of the application program. These routines assume a program can hold entire files in memory, without explicitly using temporary files. When reading a file, the library routines bring the data into memory and perform subsequent operations on the memory copy. Programs that wish to read or write large object files with this model must execute on a machine with a large process virtual address space. If the underlying operating system limits the number of open files, a program can use <code>elf_cntl(3ELF)</code> to retrieve all necessary data from the file, allowing the program to close the file descriptor and reuse it. Although the <code>elf_begin()</code> interfaces are convenient and efficient for many programs, they might be inappropriate for some. In those cases, an application may invoke the <code>elf32_xlatetom(3ELF)</code> or <code>elf32_xlatetof(3ELF)</code> data translation routines directly. These routines perform no input or output, leaving that as the application's responsibility. By assuming a larger share of the job, an application controls its input and output model.	
Library Names	Names associated with the library take several forms.	
	<code>elf_name</code>	These class-independent names perform some service, <i>name</i> , for the program.
	<code>elf32_name</code>	Service names with an embedded class, 32 here, indicate they work only for the designated class of files.
	<code>Elf_Type</code>	Data types can be class-independent as well, distinguished by <i>Type</i> .
	<code>Elf32_Type</code>	Class-dependent data types have an embedded class name, 32 here.
	<code>ELF_C_CMD</code>	Several functions take commands that control their actions. These values are members of the <code>Elf_Cmd</code> enumeration; they range from zero through <code>ELF_C_NUM-1</code> .
	<code>ELF_F_FLAG</code>	Several functions take flags that control library status and/or actions. Flags are bits that may be combined.
	<code>ELF32_FSZ_TYPE</code>	These constants give the file sizes in bytes of the basic ELF types for the 32-bit class of files. See <code>elf32_fsize()</code> for more information.
	<code>ELF_K_KIND</code>	The function <code>elf_kind()</code> identifies the <i>KIND</i> of file associated with an ELF descriptor. These values are

ELF_T_TYPE	members of the <code>Elf_Kind</code> enumeration; they range from zero through <code>ELF_K_NUM-1</code>. When a service function, such as <code>elf32_xlatetom()</code> or <code>elf32_xlatetof()</code>, deals with multiple types, names of this form specify the desired <i>TYPE</i>. Thus, for example, <code>ELF_T_EHDR</code> is directly related to <code>Elf32_Ehdr</code>. These values are members of the <code>Elf_Type</code> enumeration; they range from zero through <code>ELF_T_NUM-1</code>.

EXAMPLES**EXAMPLE 1** An interpretation of elf file.

The basic interpretation of an ELF file consists of:

- opening an ELF object file
- obtaining an ELF descriptor
- analyzing the file using the descriptor.

The following example opens the file, obtains the ELF descriptor, and prints out the names of each section in the file.

```
#include <fcntl.h>
#include <stdio.h>
#include <libelf.h>
#include <stdlib.h>
#include <string.h>
static void failure(void);
void
main(int argc, char ** argv)
{
    Elf32_Shdr *    shdr;
    Elf32_Ehdr *    ehdr;
    Elf *           elf;
    Elf_Scn *       scn;
    Elf_Data *      data;
    int             fd;
    unsigned int    cnt;

    /* Open the input file */
    if ((fd = open(argv[1], O_RDONLY)) == -1)
        exit(1);

    /* Obtain the ELF descriptor */
    (void) elf_version(EV_CURRENT);
    if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL)
        failure();

    /* Obtain the .shstrtab data buffer */
    if ((ehdr = elf32_getehdr(elf)) == NULL) ||
        ((scn = elf_getscn(elf, ehdr->e_shstrndx)) == NULL) ||
        ((data = elf_getdata(scn, NULL)) == NULL))
        failure();

    /* Traverse input filename, printing each section */
```

EXAMPLE 1 An interpretation of elf file. *(Continued)*

```
for (cnt = 1, scn = NULL; scn = elf_nextscn(elf, scn); cnt++) {
    if ((shdr = elf32_getshdr(scn)) == NULL)
        failure();
    (void) printf("[%d]    %s\n", cnt,
        (char *)data->d_buf + shdr->sh_name);
}
/* end main */

static void
failure()
{
    (void) fprintf(stderr, "%s\n", elf_errmsg(elf_errno()));
    exit(1);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO ar(3HEAD), elf32_checksum(3ELF), elf32_fsize(3ELF),
elf32_getshdr(3ELF), elf32_xlatetof(3ELF), elf_begin(3ELF),
elf_cntl(3ELF), elf_errmsg(3ELF), elf_fill(3ELF), elf_getarhdr(3ELF),
elf_getarsym(3ELF), elf_getbase(3ELF), elf_getdata(3ELF),
elf_getident(3ELF), elf_getscn(3ELF), elf_hash(3ELF), elf_kind(3ELF),
elf_memory(3ELF), elf_rawfile(3ELF), elf_strptr(3ELF), elf_update(3ELF),
elf_version(3ELF), gelf(3ELF), libelf(3LIB), attributes(5), lfcompile(5)

ANSI C Programmer's Guide

SPARC only a.out(4)

NOTES Information in the ELF headers is separated into common parts and processor-specific parts. A program can make a processor's information available by including the appropriate header: <sys/elf_NAME.h> where NAME matches the processor name as used in the ELF file header.

Name	Processor
M32	AT&T WE 32100
SPARC	SPARC
386	Intel 80386, 80486, Pentium

elf(3ELF)

Other processors will be added to the table as necessary.

To illustrate, a program could use the following code to “see” the processor-specific information for the SPARC based system.

```
#include <libelf.h>
#include <sys/elf_SPARC.h>
```

Without the `<sys/elf_SPARC.h>` definition, only the common ELF information would be visible.

A program could use the following code to “see” the processor-specific information for the Intel 80386:

```
#include <libelf.h>
#include <sys/elf_386.h>
```

Without the `<sys/elf_386.h>` definition, only the common ELF information would be visible.

Although reading the objects is rather straightforward, writing/updating them can corrupt the shared offsets among sections. Upon creation, relationships are established among the sections that must be maintained even if the object’s size is changed.

NAME	elf_begin, elf_end, elf_memory, elf_next, elf_rand – process ELF object files				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> Elf *elf_begin(int <i>fildes</i>, Elf_Cmd <i>cmd</i>, Elf *<i>ref</i>) ; int elf_end(Elf *<i>elf</i>) ; Elf *elf_memory(char *<i>image</i>, size_t <i>sz</i>) ; Elf_Cmd elf_next(Elf *<i>elf</i>) ; size_t elf_rand(Elf *<i>elf</i>, size_t <i>offset</i>) ;</pre>				
DESCRIPTION	elf_begin(), elf_end(), elf_memory(), elf_next(), and elf_rand() work together to process Executable and Linking Format (ELF) object files, either individually or as members of archives. After obtaining an ELF descriptor from elf_begin() or elf_memory(), the program may read an existing file, update an existing file, or create a new file. <i>fildes</i> is an open file descriptor that elf_begin() uses for reading or writing. <i>elf</i> is an ELF descriptor previously returned from elf_begin(). The initial file offset (see lseek(2)) is unconstrained, and the resulting file offset is undefined. <i>cmd</i> may have the following values: <table> <tr> <td>ELF_C_NULL</td><td>When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information.</td></tr> <tr> <td>ELF_C_READ</td><td>When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i>, this command examines archive members or entire files. Three cases can occur.</td></tr> </table> First, if <i>ref</i> is a null pointer, elf_begin() allocates a new ELF descriptor and prepares to process the entire file. If the file being read is an archive, elf_begin() also prepares the resulting descriptor to examine the initial archive member on the next call to elf_begin(), as if the program had used elf_next() or elf_rand() to “move” to the initial member. Second, if <i>ref</i> is a non-null descriptor associated with an archive file, elf_begin() lets a program obtain a separate ELF descriptor associated with an individual member. The program should have used elf_next() or elf_rand() to position <i>ref</i> appropriately (except for the initial member, which elf_begin() prepares; see the example below). In this case, <i>fildes</i> should be the same file descriptor used for the parent archive.	ELF_C_NULL	When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information.	ELF_C_READ	When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i> , this command examines archive members or entire files. Three cases can occur.
ELF_C_NULL	When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information.				
ELF_C_READ	When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i> , this command examines archive members or entire files. Three cases can occur.				

elf_begin(3ELF)

	Finally, if <i>ref</i> is a non-null ELF descriptor that is not an archive, <code>elf_begin()</code> increments the number of activations for the descriptor and returns <i>ref</i> , without allocating a new descriptor and without changing the descriptor's read/write permissions. To terminate the descriptor for <i>ref</i> , the program must call <code>elf_end()</code> once for each activation. See the examples below for more information.
ELF_C_RDWR	This command duplicates the actions of <code>ELF_C_READ</code> and additionally allows the program to update the file image (see <code>elf_update(3ELF)</code>). That is, using <code>ELF_C_READ</code> gives a read-only view of the file, while <code>ELF_C_RDWR</code> lets the program read <i>and</i> write the file. <code>ELF_C_RDWR</code> is not valid for archive members. If <i>ref</i> is non-null, it must have been created with the <code>ELF_C_RDWR</code> command.
ELF_C_WRITE	If the program wishes to ignore previous file contents, presumably to create a new file, it should set <i>cmd</i> to this value. <i>ref</i> is ignored for this command.

`elf_begin()` "works" on all files (including files with zero bytes), providing it can allocate memory for its internal structures and read any necessary information from the file. Programs reading object files thus may call `elf_kind(3ELF)` or `elf32_getehdr(3ELF)` to determine the file type (only object files have an ELF header). If the file is an archive with no more members to process, or an error occurs, `elf_begin()` returns a null pointer. Otherwise, the return value is a non-null ELF descriptor.

Before the first call to `elf_begin()`, a program must call `elf_version()` to coordinate versions.

`elf_end()` is used to terminate an ELF descriptor, *elf*, and to deallocate data associated with the descriptor. Until the program terminates a descriptor, the data remain allocated. A null pointer is allowed as an argument, to simplify error handling. If the program wishes to write data associated with the ELF descriptor to the file, it must use `elf_update()` before calling `elf_end()`.

Calling `elf_end()` removes one activation and returns the remaining activation count. The library does not terminate the descriptor until the activation count reaches 0. Consequently, a 0 return value indicates the ELF descriptor is no longer valid.

`elf_memory()` returns a pointer to an ELF descriptor, the ELF image has read operations enabled (`ELF_C_READ`). *image* is a pointer to an image of the Elf file mapped into memory, *sz* is the size of the ELF image. An ELF image that is mapped in with `elf_memory()` may be read and modified, but the ELF image size may not be changed.

elf_begin(3ELF)

elf_next() provides sequential access to the next archive member. That is, having an ELF descriptor, *elf*, associated with an archive member, elf_next() prepares the containing archive to access the following member when the program calls elf_begin(). After successfully positioning an archive for the next member, elf_next() returns the value ELF_C_READ. Otherwise, the open file was not an archive, *elf* was NULL, or an error occurred, and the return value is ELF_C_NULL. In either case, the return value may be passed as an argument to elf_begin(), specifying the appropriate action.

elf_rand() provides random archive processing, preparing *elf* to access an arbitrary archive member. *elf* must be a descriptor for the archive itself, not a member within the archive. *offset* gives the byte offset from the beginning of the archive to the archive header of the desired member. See elf_getarsym(3ELF) for more information about archive member offsets. When elf_rand() works, it returns *offset*. Otherwise, it returns 0, because an error occurred, *elf* was NULL, or the file was not an archive (no archive member can have a zero offset). A program may mix random and sequential archive processing.

System Services

When processing a file, the library decides when to read or write the file, depending on the program's requests. Normally, the library assumes the file descriptor remains usable for the life of the ELF descriptor. If, however, a program must process many files simultaneously and the underlying operating system limits the number of open files, the program can use elf_cntl() to let it reuse file descriptors. After calling elf_cntl() with appropriate arguments, the program may close the file descriptor without interfering with the library.

All data associated with an ELF descriptor remain allocated until elf_end() terminates the descriptor's last activation. After the descriptors have been terminated, the storage is released; attempting to reference such data gives undefined behavior. Consequently, a program that deals with multiple input (or output) files must keep the ELF descriptors active until it finishes with them.

EXAMPLES

EXAMPLE 1 A sample program of calling the elf_begin() function.

A prototype for reading a file appears on the next page. If the file is a simple object file, the program executes the loop one time, receiving a null descriptor in the second iteration. In this case, both *elf* and *arf* will have the same value, the activation count will be 2, and the program calls elf_end() twice to terminate the descriptor. If the file is an archive, the loop processes each archive member in turn, ignoring those that are not object files.

```
if (elf_version(EV_CURRENT) == EV_NONE)
{
    /* library out of date */
    /* recover from error */
}
cmd = ELF_C_READ;
arf = elf_begin(fildes, cmd, (Elf *)0);
while ((elf = elf_begin(fildes, cmd, arf)) != 0)
{
```

elf_begin(3ELF)

EXAMPLE 1 A sample program of calling the `elf_begin()` function. (Continued)

```
    if ((ehdr = elf32_getehdr(elf)) != 0)
    {
        /* process the file . . . */
    }
    cmd = elf_next(elf);
    elf_end(elf);
}
elf_end(arf);
```

Alternatively, the next example illustrates random archive processing. After identifying the file as an archive, the program repeatedly processes archive members of interest. For clarity, this example omits error checking and ignores simple object files. Additionally, this fragment preserves the ELF descriptors for all archive members, because it does not call `elf_end()` to terminate them.

```
elf_version(EV_CURRENT);
arf = elf_begin(fildes, ELF_C_READ, (Elf *)0);
if (elf_kind(arf) != ELF_K_AR)
{
    /* not an archive */
}
/* initial processing */
/* set offset = . . . for desired member header */
while (elf_rand(arf, offset) == offset)
{
    if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
        break;
    if ((ehdr = elf32_getehdr(elf)) != 0)
    {
        /* process archive member . . . */
    }
    /* set offset = . . . for desired member header */
}
```

An archive starts with a “magic string” that has SARMAG bytes; the initial archive member follows immediately. An application could thus provide the following function to rewind an archive (the function returns `-1` for errors and `0` otherwise).

```
#include <ar.h>
#include <libelf.h>
int
rewindelf(Elf *elf)
{
    if (elf_rand(elf, (size_t)SARMAG) == SARMAG)
        return 0;
    return -1;
}
```

The following outline shows how one might create a new ELF file. This example is simplified to show the overall flow.

```
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR|O_TRUNC|O_CREAT, 0666);
```

EXAMPLE 1 A sample program of calling the elf_begin() function. (Continued)

```
if ((elf = elf_begin(fildes, ELF_C_WRITE, (Elf *)0)) == 0)
    return;
ehdr = elf32_newehdr(elf);
phdr = elf32_newphdr(elf, count);
scn = elf_newscn(elf);
shdr = elf32_getshdr(scn);
data = elf_newdata(scn);
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

Finally, the following outline shows how one might update an existing ELF file. Again, this example is simplified to show the overall flow.

```
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR);
elf = elf_begin(fildes, ELF_C_RDWR, (Elf *)0);
/* add new or delete old information */
. . .
/* ensure that the memory image of the file is complete */
elf_update(elf, ELF_C_NULL);
elf_update(elf, ELF_C_WRITE); /* update file */
elf_end(elf);
```

Notice that both file creation examples open the file with write *and* read permissions. On systems that support mmap(2), the library uses it to enhance performance, and mmap(2) requires a readable file descriptor. Although the library can use a write-only file descriptor, the application will not obtain the performance advantages of mmap(2).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO creat(2), lseek(2), mmap(2), open(2), ar(3HEAD), elf(3ELF), elf32_getehdr(3ELF), elf_cntl(3ELF), elf_getarhdr(3ELF), elf_getarsym(3ELF), elf_getbase(3ELF), elf_getdata(3ELF), elf_getscn(3ELF), elf_kind(3ELF), elf_rawfile(3ELF), elf_update(3ELF), elf_version(3ELF), libelf(3LIB), attributes(5)

elf_cntl(3ELF)

NAME	elf_cntl – control an elf file descriptor						
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> int elf_cntl(Elf *elf, Elf_Cmd cmd);</pre>						
DESCRIPTION	elf_cntl() instructs the library to modify its behavior with respect to an ELF descriptor, <i>elf</i>. As elf_begin(3ELF) describes, an ELF descriptor can have multiple activations, and multiple ELF descriptors may share a single file descriptor. Generally, elf_cntl() commands apply to all activations of <i>elf</i>. Moreover, if the ELF descriptor is associated with an archive file, descriptors for members within the archive will also be affected as described below. Unless stated otherwise, operations on archive members do not affect the descriptor for the containing archive. The <i>cmd</i> argument tells what actions to take and may have the following values: <table> <tr> <td>ELF_C_FDDONE</td><td>This value tells the library not to use the file descriptor associated with <i>elf</i>. A program should use this command when it has requested all the information it cares to use and wishes to avoid the overhead of reading the rest of the file. The memory for all completed operations remains valid, but later file operations, such as the initial elf_getdata() for a section, will fail if the data are not in memory already.</td></tr> <tr> <td>ELF_C_FDREAD</td><td>This command is similar to ELF_C_FDDONE, except it forces the library to read the rest of the file. A program should use this command when it must close the file descriptor but has not yet read everything it needs from the file. After elf_cntl() completes the ELF_C_FDREAD command, future operations, such as elf_getdata(), will use the memory version of the file without needing to use the file descriptor.</td></tr> </table> If elf_cntl() succeeds, it returns 0. Otherwise <i>elf</i> was NULL or an error occurred, and the function returns -1.	ELF_C_FDDONE	This value tells the library not to use the file descriptor associated with <i>elf</i> . A program should use this command when it has requested all the information it cares to use and wishes to avoid the overhead of reading the rest of the file. The memory for all completed operations remains valid, but later file operations, such as the initial elf_getdata() for a section, will fail if the data are not in memory already.	ELF_C_FDREAD	This command is similar to ELF_C_FDDONE, except it forces the library to read the rest of the file. A program should use this command when it must close the file descriptor but has not yet read everything it needs from the file. After elf_cntl() completes the ELF_C_FDREAD command, future operations, such as elf_getdata(), will use the memory version of the file without needing to use the file descriptor.		
ELF_C_FDDONE	This value tells the library not to use the file descriptor associated with <i>elf</i> . A program should use this command when it has requested all the information it cares to use and wishes to avoid the overhead of reading the rest of the file. The memory for all completed operations remains valid, but later file operations, such as the initial elf_getdata() for a section, will fail if the data are not in memory already.						
ELF_C_FDREAD	This command is similar to ELF_C_FDDONE, except it forces the library to read the rest of the file. A program should use this command when it must close the file descriptor but has not yet read everything it needs from the file. After elf_cntl() completes the ELF_C_FDREAD command, future operations, such as elf_getdata(), will use the memory version of the file without needing to use the file descriptor.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	elf(3ELF), elf_begin(3ELF), elf_getdata(3ELF), elf_rawfile(3ELF), libelf(3LIB), attributes(5)						

elf_cntl(3ELF)

NOTES | If the program wishes to use the “raw” operations (see `elf_rawdata()`, which `elf_getdata(3ELF)` describes, and `elf_rawfile(3ELF)`) after disabling the file descriptor with `ELF_C_FDDONE` or `ELF_C_FDREAD`, it must execute the raw operations explicitly beforehand. Otherwise, the raw file operations will fail. Calling `elf_rawfile()` makes the entire image available, thus supporting subsequent `elf_rawdata()` calls.

elf_errmsg(3ELF)

NAME	elf_errmsg, elf_errno – error handling						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> const char *elf_errmsg(int <i>err</i>); int elf_errno(void);</pre>						
DESCRIPTION	If an ELF library function fails, a program can call <code>elf_errno()</code> to retrieve the library's internal error number. As a side effect, this function resets the internal error number to 0, which indicates no error. The <code>elf_errmsg()</code> function takes an error number, <i>err</i>, and returns a null-terminated error message (with no trailing new-line) that describes the problem. A zero <i>err</i> retrieves a message for the most recent error. If no error has occurred, the return value is a null pointer (not a pointer to the null string). Using <i>err</i> of -1 also retrieves the most recent error, except it guarantees a non-null return value, even when no error has occurred. If no message is available for the given number, <code>elf_errmsg()</code> returns a pointer to an appropriate message. This function does not have the side effect of clearing the internal error number.						
EXAMPLES	EXAMPLE 1 A sample program of calling the <code>elf_errmsg()</code> function. The following fragment clears the internal error number and checks it later for errors. Unless an error occurs after the first call to <code>elf_errno()</code>, the next call will return 0. <pre>(void)elf_errno(); /* processing . . . */ while (more_to_do) { if ((err = elf_errno()) != 0) { /* print msg */ msg = elf_errmsg(err); } }</pre>						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						

NAME	elf_fill – set fill byte						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> void elf_fill(int <i>fill</i>);</pre>						
DESCRIPTION	Alignment constraints for ELF files sometimes require the presence of “holes.” For example, if the data for one section are required to begin on an eight-byte boundary, but the preceding section is too “short,” the library must fill the intervening bytes. These bytes are set to the <i>fill</i> character. The library uses zero bytes unless the application supplies a value. See elf_getdata(3ELF) for more information about these holes.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	elf(3ELF), elf_flagdata(3ELF), elf_getdata(3ELF), elf_update(3ELF), libelf(3LIB), attributes(5)						
NOTES	An application can assume control of the object file organization by setting the ELF_F_LAYOUT bit (see elf_flagdata(3ELF)). When this is done, the library does <i>not</i> fill holes.						

elf_flagdata(3ELF)

NAME	elf_flagdata, elf_flagehdr, elf_flagelf, elf_flagphdr, elf_flagscn, elf_flagshdr – manipulate flags								
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> unsigned elf_flagdata(Elf_Data *<i>data</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagehdr(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagelf(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagphdr(Elf *<i>elf</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagscn(Elf_Scn *<i>scn</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>); unsigned elf_flagshdr(Elf_Scn *<i>scn</i>, Elf_Cmd <i>cmd</i>, unsigned <i>flags</i>);</pre>								
DESCRIPTION	These functions manipulate the flags associated with various structures of an ELF file. Given an ELF descriptor (<i>elf</i>), a data descriptor (<i>data</i>), or a section descriptor (<i>scn</i>), the functions may set or clear the associated status bits, returning the updated bits. A null descriptor is allowed, to simplify error handling; all functions return 0 for this degenerate case. <i>cmd</i> may have the following values: <table> <tr> <td>ELF_C_CLR</td><td>The functions clear the bits that are asserted in <i>flags</i>. Only the non-zero bits in <i>flags</i> are cleared; zero bits do not change the status of the descriptor.</td></tr> <tr> <td>ELF_C_SET</td><td>The functions set the bits that are asserted in <i>flags</i>. Only the non-zero bits in <i>flags</i> are set; zero bits do not change the status of the descriptor.</td></tr> </table> Descriptions of the defined <i>flags</i> bits appear below: <table> <tr> <td>ELF_F_DIRTY</td><td>When the program intends to write an ELF file, this flag asserts the associated information needs to be written to the file. Thus, for example, a program that wished to update the ELF header of an existing file would call <code>elf_flagehdr()</code> with this bit set in <i>flags</i> and <i>cmd</i> equal to <code>ELF_C_SET</code>. A later call to <code>elf_update()</code> would write the marked header to the file.</td></tr> <tr> <td>ELF_F_LAYOUT</td><td>Normally, the library decides how to arrange an output file. That is, it automatically decides where to place sections, how to align them in the file, etc. If this bit is set for an ELF descriptor, the program assumes responsibility for determining all file positions. This bit is meaningful only for <code>elf_flagelf()</code> and applies to the entire file associated with the descriptor.</td></tr> </table> When a flag bit is set for an item, it affects all the subitems as well. Thus, for example, if the program sets the <code>ELF_F_DIRTY</code> bit with <code>elf_flagelf()</code>, the entire logical file is “dirty.”	ELF_C_CLR	The functions clear the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are cleared; zero bits do not change the status of the descriptor.	ELF_C_SET	The functions set the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are set; zero bits do not change the status of the descriptor.	ELF_F_DIRTY	When the program intends to write an ELF file, this flag asserts the associated information needs to be written to the file. Thus, for example, a program that wished to update the ELF header of an existing file would call <code>elf_flagehdr()</code> with this bit set in <i>flags</i> and <i>cmd</i> equal to <code>ELF_C_SET</code> . A later call to <code>elf_update()</code> would write the marked header to the file.	ELF_F_LAYOUT	Normally, the library decides how to arrange an output file. That is, it automatically decides where to place sections, how to align them in the file, etc. If this bit is set for an ELF descriptor, the program assumes responsibility for determining all file positions. This bit is meaningful only for <code>elf_flagelf()</code> and applies to the entire file associated with the descriptor.
ELF_C_CLR	The functions clear the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are cleared; zero bits do not change the status of the descriptor.								
ELF_C_SET	The functions set the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are set; zero bits do not change the status of the descriptor.								
ELF_F_DIRTY	When the program intends to write an ELF file, this flag asserts the associated information needs to be written to the file. Thus, for example, a program that wished to update the ELF header of an existing file would call <code>elf_flagehdr()</code> with this bit set in <i>flags</i> and <i>cmd</i> equal to <code>ELF_C_SET</code> . A later call to <code>elf_update()</code> would write the marked header to the file.								
ELF_F_LAYOUT	Normally, the library decides how to arrange an output file. That is, it automatically decides where to place sections, how to align them in the file, etc. If this bit is set for an ELF descriptor, the program assumes responsibility for determining all file positions. This bit is meaningful only for <code>elf_flagelf()</code> and applies to the entire file associated with the descriptor.								

EXAMPLES **EXAMPLE 1** A sample display of calling the `elf_flagdata()` function.

The following fragment shows how one might mark the ELF header to be written to the output file:

```
/* dirty ehdr . . . */
ehdr = elf32_getehdr(elf);
elf_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO `elf(3ELF)`, `elf32_getehdr(3ELF)`, `elf_getdata(3ELF)`, `elf_update(3ELF)`, `attributes(5)`

elf_getarhdr(3ELF)

NAME	elf_getarhdr – retrieve archive member header
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i>...] #include <libelf.h> Elf_Arhdr *elf_getarhdr(Elf *elf);</pre>
DESCRIPTION	elf_getarhdr() returns a pointer to an archive member header, if one is available for the ELF descriptor <i>elf</i>. Otherwise, no archive member header exists, an error occurred, or <i>elf</i> was null; elf_getarhdr() then returns a null value. The header includes the following members. <pre>char *ar_name; time_t ar_date; uid_t ar_uid; gid_t ar_gid; mode_t ar_mode; off_t ar_size; char *ar_rawname;</pre> An archive member name, available through <i>ar_name</i>, is a null-terminated string, with the ar format control characters removed. The <i>ar_rawname</i> member holds a null-terminated string that represents the original name bytes in the file, including the terminating slash and trailing blanks as specified in the archive format. In addition to “regular” archive members, the archive format defines some special members. All special member names begin with a slash (/), distinguishing them from regular members (whose names may not contain a slash). These special members have the names (<i>ar_name</i>) defined below. / This is the archive symbol table. If present, it will be the first archive member. A program may access the archive symbol table through elf_getarsym(). The information in the symbol table is useful for random archive processing (see elf_rand() on elf_begin(3ELF)). // This member, if present, holds a string table for long archive member names. An archive member’s header contains a 16-byte area for the name, which may be exceeded in some file systems. The library automatically retrieves long member names from the string table, setting <i>ar_name</i> to the appropriate value. Under some error conditions, a member’s name might not be available. Although this causes the library to set <i>ar_name</i> to a null pointer, the <i>ar_rawname</i> member will be set as usual.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable

elf_getarhdr(3ELF)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO ar(3HEAD), elf(3ELF), elf_begin(3ELF), elf_getarsym(3ELF), libelf(3LIB), attributes(5)

elf_getarsym(3ELF)

NAME	elf_getarsym – retrieve archive symbol table						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> Elf_Arsym *elf_getarsym(Elf *elf, size_t *ptr);</pre>						
DESCRIPTION	The <code>elf_getarsym()</code> function returns a pointer to the archive symbol table, if one is available for the ELF descriptor <i>elf</i>. Otherwise, the archive doesn't have a symbol table, an error occurred, or <i>elf</i> was null; <code>elf_getarsym()</code> then returns a null value. The symbol table is an array of structures that include the following members. <pre>char *as_name; size_t as_off; unsigned long as_hash;</pre> These members have the following semantics: <table><tr><td>as_name</td><td>A pointer to a null-terminated symbol name resides here.</td></tr><tr><td>as_off</td><td>This value is a byte offset from the beginning of the archive to the member's header. The archive member residing at the given offset defines the associated symbol. Values in <code>as_off</code> may be passed as arguments to <code>elf_rand()</code>. See <code>elf_begin(3ELF)</code> to access the desired archive member.</td></tr><tr><td>as_hash</td><td>This is a hash value for the name, as computed by <code>elf_hash()</code>.</td></tr></table> If <i>ptr</i> is non-null, the library stores the number of table entries in the location to which <i>ptr</i> points. This value is set to 0 when the return value is NULL. The table's last entry, which is included in the count, has a null <code>as_name</code>, a zero value for <code>as_off</code>, and <code>~0UL</code> for <code>as_hash</code>. The hash value returned is guaranteed not to be the bit pattern of all ones (<code>~0UL</code>).	as_name	A pointer to a null-terminated symbol name resides here.	as_off	This value is a byte offset from the beginning of the archive to the member's header. The archive member residing at the given offset defines the associated symbol. Values in <code>as_off</code> may be passed as arguments to <code>elf_rand()</code> . See <code>elf_begin(3ELF)</code> to access the desired archive member.	as_hash	This is a hash value for the name, as computed by <code>elf_hash()</code> .
as_name	A pointer to a null-terminated symbol name resides here.						
as_off	This value is a byte offset from the beginning of the archive to the member's header. The archive member residing at the given offset defines the associated symbol. Values in <code>as_off</code> may be passed as arguments to <code>elf_rand()</code> . See <code>elf_begin(3ELF)</code> to access the desired archive member.						
as_hash	This is a hash value for the name, as computed by <code>elf_hash()</code> .						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>ar(3HEAD)</code> , <code>elf(3ELF)</code> , <code>elf_begin(3ELF)</code> , <code>elf_getarhdr(3ELF)</code> , <code>elf_hash(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						

NAME	elf_getbase – get the base offset for an object file						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> off_t elf_getbase(Elf *<i>elf</i>) ;</pre>						
DESCRIPTION	The <code>elf_getbase()</code> function returns the file offset of the first byte of the file or archive member associated with <i>elf</i>, if it is known or obtainable, and <code>-1</code> otherwise. A null <i>elf</i> is allowed, to simplify error handling; the return value in this case is <code>-1</code>. The base offset of an archive member is the beginning of the member's information, <i>not</i> the beginning of the archive member header.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>ar(3HEAD)</code> , <code>elf(3ELF)</code> , <code>elf_begin(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						

elf_getdata(3ELF)

NAME	elf_getdata, elf_newdata, elf_rawdata – get section data
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l~~elf~~ [<i>library</i> ...] #include <lib~~elf~~.h> Elf_Data *elf_getdata(Elf_Scn *scn, Elf_Data *data); Elf_Data *elf_newdata(Elf_Scn *scn); Elf_Data *elf_rawdata(Elf_Scn *scn, Elf_Data *data);</pre>
DESCRIPTION	These functions access and manipulate the data associated with a section descriptor, <i>scn</i>. When reading an existing file, a section will have a single data buffer associated with it. A program may build a new section in pieces, however, composing the new data from multiple data buffers. For this reason, the data for a section should be viewed as a list of buffers, each of which is available through a data descriptor. The <code>elf_getdata()</code> function lets a program step through a section's data list. If the incoming data descriptor, <i>data</i>, is null, the function returns the first buffer associated with the section. Otherwise, <i>data</i> should be a data descriptor associated with <i>scn</i>, and the function gives the program access to the next data element for the section. If <i>scn</i> is null or an error occurs, <code>elf_getdata()</code> returns a null pointer. The <code>elf_getdata()</code> function translates the data from file representations into memory representations (see <code>elf32_xlatetof(3ELF)</code>) and presents objects with memory data types to the program, based on the file's <i>class</i> (see <code>elf(3ELF)</code>). The working library version (see <code>elf_version(3ELF)</code>) specifies what version of the memory structures the program wishes <code>elf_getdata()</code> to present. The <code>elf_newdata()</code> function creates a new data descriptor for a section, appending it to any data elements already associated with the section. As described below, the new data descriptor appears empty, indicating the element holds no data. For convenience, the descriptor's type (<i>d_type</i> below) is set to <code>ELF_T_BYTE</code>, and the version (<i>d_version</i> below) is set to the working version. The program is responsible for setting (or changing) the descriptor members as needed. This function implicitly sets the <code>ELF_F_DIRTY</code> bit for the section's data (see <code>elf_flagdata(3ELF)</code>). If <i>scn</i> is null or an error occurs, <code>elf_newdata()</code> returns a null pointer. The <code>elf_rawdata()</code> function differs from <code>elf_getdata()</code> by returning only uninterpreted bytes, regardless of the section type. This function typically should be used only to retrieve a section image from a file being read, and then only when a program must avoid the automatic data translation described below. Moreover, a program may not close or disable (see <code>elf_cntl(3ELF)</code>) the file descriptor associated with <i>elf</i> before the initial raw operation, because <code>elf_rawdata()</code> might read the data from the file to ensure it doesn't interfere with <code>elf_getdata()</code>. See <code>elf_rawfile(3ELF)</code> for a related facility that applies to the entire file. When <code>elf_getdata()</code> provides the right translation, its use is recommended over <code>elf_rawdata()</code>. If <i>scn</i> is null or an error occurs, <code>elf_rawdata()</code> returns a null pointer. The <code>Elf_Data</code> structure includes the following members:

```

void      *d_buf;
Elf_Type  d_type;
size_t    d_size;
off_t     d_off;
size_t    d_align;
unsigned   d_version;

```

These members are available for direct manipulation by the program. Descriptions appear below.

<code>d_buf</code>	A pointer to the data buffer resides here. A data element with no data has a null pointer.
<code>d_type</code>	This member's value specifies the type of the data to which <code>d_buf</code> points. A section's type determines how to interpret the section contents, as summarized below.
<code>d_size</code>	This member holds the total size, in bytes, of the memory occupied by the data. This may differ from the size as represented in the file. The size will be zero if no data exist. (See the discussion of <code>SHT_NOBITS</code> below for more information.)
<code>d_off</code>	This member gives the offset, within the section, at which the buffer resides. This offset is relative to the file's section, not the memory object's.
<code>d_align</code>	This member holds the buffer's required alignment, from the beginning of the section. That is, <code>d_off</code> will be a multiple of this member's value. For example, if this member's value is 4, the beginning of the buffer will be four-byte aligned within the section. Moreover, the entire section will be aligned to the maximum of its constituents, thus ensuring appropriate alignment for a buffer within the section and within the file.
<code>d_version</code>	This member holds the version number of the objects in the buffer. When the library originally read the data from the object file, it used the working version to control the translation to memory objects.

Data Alignment As mentioned above, data buffers within a section have explicit alignment constraints. Consequently, adjacent buffers sometimes will not abut, causing "holes" within a section. Programs that create output files have two ways of dealing with these holes.

First, the program can use `elf_fill()` to tell the library how to set the intervening bytes. When the library must generate gaps in the file, it uses the fill byte to initialize the data there. The library's initial fill value is 0, and `elf_fill()` lets the application change that.

elf_getdata(3ELF)

Section and Memory Types

Second, the application can generate its own data buffers to occupy the gaps, filling the gaps with values appropriate for the section being created. A program might even use different fill values for different sections. For example, it could set text sections' bytes to no-operation instructions, while filling data section holes with zero. Using this technique, the library finds no holes to fill, because the application eliminated them.

The `elf_getdata()` function interprets sections' data according to the section type, as noted in the section header available through `elf32_getshdr()`. The following table shows the section types and how the library represents them with memory data types for the 32-bit file class. Other classes would have similar tables. By implication, the memory data types control translation by `elf32_xlatetof(3ELF)`

Section Type	Elf_Type	32-bit Type
SHT_DYNAMIC	ELF_T_DYN	Elf32_Dyn
SHT_DYNSYM	ELF_T_SYM	Elf32_Sym
SHT_FINI_ARRAY	ELF_T_ADDR	Elf32_Addr
SHT_GROUP	ELF_T_WORD	Elf32_Word
SHT_HASH	ELF_T_WORD	Elf32_Word
SHT_INIT_ARRAY	ELF_T_ADDR	Elf32_Addr
SHT_NOBITS	ELF_T_BYTE	unsigned char
SHT_NOTE	ELF_T_NOTE	unsigned char
SHT_NULL	<i>none</i>	<i>none</i>
SHT_PREINIT_ARRAY	ELF_T_ADDR	Elf32_Addr
SHT_PROGBITS	ELF_T_BYTE	unsigned char
SHT_REL	ELF_T_REL	Elf32_Rel
SHT_RELA	ELF_T_RELA	Elf32_Rela
SHT_STRTAB	ELF_T_BYTE	unsigned char
SHT_SYMTAB	ELF_T_SYM	Elf32_Sym
SHT_SUNW_comdat	ELF_T_BYTE	unsigned char
SHT_SUNW_move	ELF_T_MOVE	Elf32_Move (sparc)
SHT_SUNW_move	ELF_T_MOVEP	Elf32_Move (ia32)
SHT_SUNW_syminfo	ELF_T_SYMINFO	Elf32_Syminfo
SHT_SUNW_verdef	ELF_T_VDEF	Elf32_Verdef
SHT_SUNW_verneed	ELF_T_VNEED	Elf32_Verneed

Section Type	Elf_Type	32-bit Type
SHT_SUNW_versym	ELF_T_HALF	Elf32_Versym
<i>other</i>	ELF_T_BYTE	unsigned char

The `elf_rawdata()` function creates a buffer with type `ELF_T_BYTE`.

As mentioned above, the program's working version controls what structures the library creates for the application. The library similarly interprets section types according to the versions. If a section type belongs to a version newer than the application's working version, the library does not translate the section data. Because the application cannot know the data format in this case, the library presents an untranslated buffer of type `ELF_T_BYTE`, just as it would for an unrecognized section type.

A section with a special type, `SHT_NOBITS`, occupies no space in an object file, even when the section header indicates a non-zero size. `elf_getdata()` and `elf_rawdata()` work on such a section, setting the `data` structure to have a null buffer pointer and the type indicated above. Although no data are present, the `d_size` value is set to the size from the section header. When a program is creating a new section of type `SHT_NOBITS`, it should use `elf_newdata()` to add data buffers to the section. These empty data buffers should have the `d_size` members set to the desired size and the `d_buf` members set to `NULL`.

EXAMPLES

EXAMPLE 1 A sample program of calling `elf_getdata()`.

The following fragment obtains the string table that holds section names (ignoring error checking). See `elf_strptr(3ELF)` for a variation of string table handling.

```
ehdr = elf32_getehdr(elf);
scn = elf_getscn(elf, (size_t)ehdr->e_shstrndx);
shdr = elf32_getshdr(scn);
if (shdr->sh_type != SHT_STRTAB)
{
    /* not a string table */
}
data = 0;
if ((data = elf_getdata(scn, data)) == 0 || data->d_size == 0)
{
    /* error or no data */
}
```

The `e_shstrndx` member in an ELF header holds the section table index of the string table. The program gets a section descriptor for that section, verifies it is a string table, and then retrieves the data. When this fragment finishes, `data->d_buf` points at the first byte of the string table, and `data->d_size` holds the string table's size in bytes.

elf_getdata(3ELF)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO

elf(3ELF), elf32_getehdr(3ELF), elf64_getehdr(3ELF),
elf32_getshdr(3ELF), elf64_getshdr(3ELF), elf32_xlatetof(3ELF),
elf64_xlatetof(3ELF), elf_cntl(3ELF), elf_fill(3ELF),
elf_flagdata(3ELF), elf_getscn(3ELF), elf_rawfile(3ELF),
elf_strptr(3ELF), elf_version(3ELF), libelf(3LIB), attributes(5)

NAME	elf_getident – retrieve file identification data																												
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> char *elf_getident(Elf *elf, size_t *ptr);</pre>																												
DESCRIPTION	As elf(3ELF) explains, ELF provides a framework for various classes of files, where basic objects may have 32 bits, 64 bits, etc. To accommodate these differences, without forcing the larger sizes on smaller machines, the initial bytes in an ELF file hold identification information common to all file classes. Every ELF header’s e_ident has EI_NIDENT bytes with the following interpretation: <table><tr><th>e_ident Index</th><th>Value</th><th>Purpose</th></tr><tr><td>EI_MAG0</td><td>ELFMAG0</td><td rowspan="4">File identification</td></tr><tr><td>EI_MAG1</td><td>ELFMAG1</td></tr><tr><td>EI_MAG2</td><td>ELFMAG2</td></tr><tr><td>EI_MAG3</td><td>ELFMAG3</td></tr><tr><td rowspan="3">EI_CLASS</td><td>ELFCLASSNONE</td><td rowspan="3">File class</td></tr><tr><td>ELFCLASS32</td></tr><tr><td>ELFCLASS64</td></tr><tr><td rowspan="3">EI_DATA</td><td>ELFDATANONE</td><td rowspan="3">Data encoding</td></tr><tr><td>ELFDATA2LSB</td></tr><tr><td>ELFDATA2MSB</td></tr><tr><td>EI_VERSION</td><td>EV_CURRENT</td><td>File version</td></tr><tr><td>7-15</td><td>0</td><td>Unused, set to zero</td></tr></table> Other kinds of files (see elf_kind(3ELF)) also may have identification data, though they would not conform to e_ident. elf_getident() returns a pointer to the file’s “initial bytes.” If the library recognizes the file, a conversion from the file image to the memory image may occur. In any case, the identification bytes are guaranteed not to have been modified, though	e_ident Index	Value	Purpose	EI_MAG0	ELFMAG0	File identification	EI_MAG1	ELFMAG1	EI_MAG2	ELFMAG2	EI_MAG3	ELFMAG3	EI_CLASS	ELFCLASSNONE	File class	ELFCLASS32	ELFCLASS64	EI_DATA	ELFDATANONE	Data encoding	ELFDATA2LSB	ELFDATA2MSB	EI_VERSION	EV_CURRENT	File version	7-15	0	Unused, set to zero
e_ident Index	Value	Purpose																											
EI_MAG0	ELFMAG0	File identification																											
EI_MAG1	ELFMAG1																												
EI_MAG2	ELFMAG2																												
EI_MAG3	ELFMAG3																												
EI_CLASS	ELFCLASSNONE	File class																											
	ELFCLASS32																												
	ELFCLASS64																												
EI_DATA	ELFDATANONE	Data encoding																											
	ELFDATA2LSB																												
	ELFDATA2MSB																												
EI_VERSION	EV_CURRENT	File version																											
7-15	0	Unused, set to zero																											

elf_getident(3ELF)

the size of the unmodified area depends on the file type. If *ptr* is non-null, the library stores the number of identification bytes in the location to which *ptr* points. If no data are present, *elf* is null, or an error occurs, the return value is a null pointer, with 0 stored through *ptr*, if *ptr* is non-null.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO `elf(3ELF)`, `elf32_getehdr(3ELF)`, `elf_begin(3ELF)`, `elf_kind(3ELF)`, `elf_rawfile(3ELF)`, `libelf(3LIB)`, `attributes(5)`

NAME	elf_getscn, elf_ndxscn, elf_newscn, elf_nextscn – get section information
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> Elf_Scn *elf_getscn(Elf *elf, size_t index); size_t elf_ndxscn(Elf_Scn *scn); Elf_Scn *elf_newscn(Elf *elf); Elf_Scn *elf_nextscn(Elf *elf, Elf_Scn *scn);</pre>
DESCRIPTION	These functions provide indexed and sequential access to the sections associated with the ELF descriptor <i>elf</i>. If the program is building a new file, it is responsible for creating the file's ELF header before creating sections; see elf32_getehdr(3ELF). The elf_getscn() function returns a section descriptor, given an <i>index</i> into the file's section header table. Note that the first “real” section has an index of 1. Although a program can get a section descriptor for the section whose <i>index</i> is 0 (SHN_UNDEF, the undefined section), the section has no data and the section header is “empty” (though present). If the specified section does not exist, an error occurs, or <i>elf</i> is NULL, elf_getscn() returns a null pointer. The elf_newscn() function creates a new section and appends it to the list for <i>elf</i>. Because the SHN_UNDEF section is required and not “interesting” to applications, the library creates it automatically. Thus the first call to elf_newscn() for an ELF descriptor with no existing sections returns a descriptor for section 1. If an error occurs or <i>elf</i> is NULL, elf_newscn() returns a null pointer. After creating a new section descriptor, the program can use elf32_getshdr() to retrieve the newly created, “clean” section header. The new section descriptor will have no associated data (see elf_getdata(3ELF)). When creating a new section in this way, the library updates the e_shnum member of the ELF header and sets the ELF_F_DIRTY bit for the section (see elf_flagdata(3ELF)). If the program is building a new file, it is responsible for creating the file's ELF header (see elf32_getehdr(3ELF)) before creating new sections. The elf_nextscn() function takes an existing section descriptor, <i>scn</i>, and returns a section descriptor for the next higher section. One may use a null <i>scn</i> to obtain a section descriptor for the section whose index is 1 (skipping the section whose index is SHN_UNDEF). If no further sections are present or an error occurs, elf_nextscn() returns a null pointer. The elf_ndxscn() function takes an existing section descriptor, <i>scn</i>, and returns its section table index. If <i>scn</i> is null or an error occurs, elf_ndxscn() returns SHN_UNDEF.
EXAMPLES	EXAMPLE 1 A sample of calling elf_getscn() function. An example of sequential access appears below. Each pass through the loop processes the next section in the file; the loop terminates when all sections have been processed.

elf_getscn(3ELF)

EXAMPLE 1 A sample of calling elf_getscn() function. *(Continued)*

```
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
    /* process section */
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO elf(3ELF), elf32_getehdr(3ELF), elf32_getshdr(3ELF), elf_begin(3ELF), elf_flagdata(3ELF), elf_getdata(3ELF), libelf(3LIB), attributes(5)

NAME	elf_hash – compute hash value						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> unsigned long elf_hash(const char *<i>name</i>);</pre>						
DESCRIPTION	The <code>elf_hash()</code> function computes a hash value, given a null terminated string, <i>name</i>. The returned hash value, <i>h</i>, can be used as a bucket index, typically after computing $h \bmod x$ to ensure appropriate bounds. Hash tables may be built on one machine and used on another because <code>elf_hash()</code> uses unsigned arithmetic to avoid possible differences in various machines' signed arithmetic. Although <i>name</i> is shown as <code>char*</code> above, <code>elf_hash()</code> treats it as unsigned <code>char*</code> to avoid sign extension differences. Using <code>char*</code> eliminates type conflicts with expressions such as <code>elf_hash(name)</code>. ELF files' symbol hash tables are computed using this function (see <code>elf_getdata(3ELF)</code> and <code>elf32_xlatetof(3ELF)</code>). The hash value returned is guaranteed not to be the bit pattern of all ones (<code>~0UL</code>).						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>elf32_xlatetof(3ELF)</code> , <code>elf_getdata(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						

elf_kind(3ELF)

NAME	elf_kind – determine file type								
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf_Kind elf_kind(Elf *elf);</pre>								
DESCRIPTION	This function returns a value identifying the kind of file associated with an ELF descriptor (<i>elf</i>). Defined values are below: <table><tr><td>ELF_K_AR</td><td>The file is an archive [see ar(3HEAD)]. An ELF descriptor may also be associated with an archive <i>member</i>, not the archive itself, and then <code>elf_kind()</code> identifies the member's type.</td></tr><tr><td>ELF_K_COFF</td><td>The file is a COFF object file. <code>elf_begin(3ELF)</code> describes the library's handling for COFF files.</td></tr><tr><td>ELF_K_ELF</td><td>The file is an ELF file. The program may use <code>elf_getident()</code> to determine the class. Other functions, such as <code>elf32_getehdr()</code>, are available to retrieve other file information.</td></tr><tr><td>ELF_K_NONE</td><td>This indicates a kind of file unknown to the library.</td></tr></table> Other values are reserved, to be assigned as needed to new kinds of files. <i>elf</i> should be a value previously returned by <code>elf_begin()</code>. A null pointer is allowed, to simplify error handling, and causes <code>elf_kind()</code> to return <code>ELF_K_NONE</code>.	ELF_K_AR	The file is an archive [see ar(3HEAD)]. An ELF descriptor may also be associated with an archive <i>member</i> , not the archive itself, and then <code>elf_kind()</code> identifies the member's type.	ELF_K_COFF	The file is a COFF object file. <code>elf_begin(3ELF)</code> describes the library's handling for COFF files.	ELF_K_ELF	The file is an ELF file. The program may use <code>elf_getident()</code> to determine the class. Other functions, such as <code>elf32_getehdr()</code> , are available to retrieve other file information.	ELF_K_NONE	This indicates a kind of file unknown to the library.
ELF_K_AR	The file is an archive [see ar(3HEAD)]. An ELF descriptor may also be associated with an archive <i>member</i> , not the archive itself, and then <code>elf_kind()</code> identifies the member's type.								
ELF_K_COFF	The file is a COFF object file. <code>elf_begin(3ELF)</code> describes the library's handling for COFF files.								
ELF_K_ELF	The file is an ELF file. The program may use <code>elf_getident()</code> to determine the class. Other functions, such as <code>elf32_getehdr()</code> , are available to retrieve other file information.								
ELF_K_NONE	This indicates a kind of file unknown to the library.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Stable								
MT-Level	MT-Safe								
SEE ALSO	<code>ar(3HEAD)</code> , <code>elf(3ELF)</code> , <code>elf32_getehdr(3ELF)</code> , <code>elf_begin(3ELF)</code> , <code>elf_getident(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>								

NAME	elf_rawfile – retrieve uninterpreted file contents						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file</i> ... -l<i>elf</i> [<i>library</i> ...] #include <libelf.h> char *elf_rawfile(Elf *elf, size_t *ptr);</pre>						
DESCRIPTION	The <code>elf_rawfile()</code> function returns a pointer to an uninterpreted byte image of the file. This function should be used only to retrieve a file being read. For example, a program might use <code>elf_rawfile()</code> to retrieve the bytes for an archive member. A program may not close or disable (see <code>elf_cntl(3ELF)</code>) the file descriptor associated with <i>elf</i> before the initial call to <code>elf_rawfile()</code>, because <code>elf_rawfile()</code> might have to read the data from the file if it does not already have the original bytes in memory. Generally, this function is more efficient for unknown file types than for object files. The library implicitly translates object files in memory, while it leaves unknown files unmodified. Thus, asking for the uninterpreted image of an object file may create a duplicate copy in memory. <code>elf_rawdata()</code> is a related function, providing access to sections within a file. See <code>elf_getdata(3ELF)</code>. If <i>ptr</i> is non-null, the library also stores the file's size, in bytes, in the location to which <i>ptr</i> points. If no data are present, <i>elf</i> is null, or an error occurs, the return value is a null pointer, with 0 stored through <i>ptr</i>, if <i>ptr</i> is non-null.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>elf32_getehdr(3ELF)</code> , <code>elf_begin(3ELF)</code> , <code>elf_cntl(3ELF)</code> , <code>elf_getdata(3ELF)</code> , <code>elf_getident(3ELF)</code> , <code>elf_kind(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						
NOTES	A program that uses <code>elf_rawfile()</code> and that also interprets the same file as an object file potentially has two copies of the bytes in memory. If such a program requests the raw image first, before it asks for translated information (through such functions as <code>elf32_getehdr()</code>, <code>elf_getdata()</code>, and so on), the library “freezes” its original memory copy for the raw image. It then uses this frozen copy as the source for creating translated objects, without reading the file again. Consequently, the application should view the raw file image returned by <code>elf_rawfile()</code> as a read-only buffer, unless it wants to alter its own view of data subsequently translated. In any case, the application may alter the translated objects without changing bytes visible in the raw image.						

`elf_rawfile(3ELF)`

Multiple calls to `elf_rawfile()` with the same ELF descriptor return the same value; the library does not create duplicate copies of the file.

NAME	elf_strptr – make a string pointer						
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> char *elf_strptr(Elf *elf, size_t section, size_t offset);</pre>						
DESCRIPTION	The <code>elf_strptr()</code> function converts a string section <i>offset</i> to a string pointer. <i>elf</i> identifies the file in which the string section resides, and <i>section</i> identifies the section table index for the strings. <code>elf_strptr()</code> normally returns a pointer to a string, but it returns a null pointer when <i>elf</i> is null, <i>section</i> is invalid or is not a section of type <code>SHT_STRTAB</code>, the section data cannot be obtained, <i>offset</i> is invalid, or an error occurs.						
EXAMPLES	EXAMPLE 1 A sample program of calling <code>elf_strptr()</code> function. A prototype for retrieving section names appears below. The file header specifies the section name string table in the <code>e_shstrndx</code> member. The following code loops through the sections, printing their names. <pre>/* handle the error */ if ((ehdr = elf32_getehdr(elf)) == 0) { return; } ndx = ehdr->e_shstrndx; scn = 0; while ((scn = elf_nextscn(elf, scn)) != 0) { char *name = 0; if ((shdr = elf32_getshdr(scn)) != 0) name = elf_strptr(elf, ndx, (size_t)shdr->sh_name); printf("%s'\n", name? name: "(null)"); }</pre>						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>elf32_getshdr(3ELF)</code> , <code>elf32_xlatetof(3ELF)</code> , <code>elf_getdata(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						
NOTES	A program may call <code>elf_getdata()</code> to retrieve an entire string table section. For some applications, that would be both more efficient and more convenient than using <code>elf_strptr()</code>.						

elf_update(3ELF)

NAME	elf_update – update an ELF descriptor						
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> off_t elf_update(Elf *elf, Elf_Cmd cmd);</pre>						
DESCRIPTION	The <code>elf_update()</code> function causes the library to examine the information associated with an ELF descriptor, <i>elf</i>, and to recalculate the structural data needed to generate the file's image. The <i>cmd</i> argument can have the following values: <table><tr><td>ELF_C_NULL</td><td>This value tells <code>elf_update()</code> to recalculate various values, updating only the ELF descriptor's memory structures. Any modified structures are flagged with the <code>ELF_F_DIRTY</code> bit. A program thus can update the structural information and then reexamine them without changing the file associated with the ELF descriptor. Because this does not change the file, the ELF descriptor may allow reading, writing, or both reading and writing (see <code>elf_begin(3ELF)</code>).</td></tr><tr><td>ELF_C_WRITE</td><td>If <i>cmd</i> has this value, <code>elf_update()</code> duplicates its <code>ELF_C_NULL</code> actions and also writes any "dirty" information associated with the ELF descriptor to the file. That is, when a program has used <code>elf_getdata(3ELF)</code> or the <code>elf_flagdata(3ELF)</code> facilities to supply new (or update existing) information for an ELF descriptor, those data will be examined, coordinated, translated if necessary (see <code>elf32_xlatetof(3ELF)</code>), and written to the file. When portions of the file are written, any <code>ELF_F_DIRTY</code> bits are reset, indicating those items no longer need to be written to the file (see <code>elf_flagdata(3ELF)</code>). The sections' data are written in the order of their section header entries, and the section header table is written to the end of the file. When the ELF descriptor was created with <code>elf_begin()</code>, it must have allowed writing the file. That is, the <code>elf_begin()</code> command must have been either <code>ELF_C_RDWR</code> or <code>ELF_C_WRITE</code>.</td></tr></table> If <code>elf_update()</code> succeeds, it returns the total size of the file image (not the memory image), in bytes. Otherwise an error occurred, and the function returns <code>-1</code>. When updating the internal structures, <code>elf_update()</code> sets some members itself. Members listed below are the application's responsibility and retain the values given by the program. The following table shows ELF Header members: <table><tr><th>Member</th><th>Notes</th></tr></table>	ELF_C_NULL	This value tells <code>elf_update()</code> to recalculate various values, updating only the ELF descriptor's memory structures. Any modified structures are flagged with the <code>ELF_F_DIRTY</code> bit. A program thus can update the structural information and then reexamine them without changing the file associated with the ELF descriptor. Because this does not change the file, the ELF descriptor may allow reading, writing, or both reading and writing (see <code>elf_begin(3ELF)</code>).	ELF_C_WRITE	If <i>cmd</i> has this value, <code>elf_update()</code> duplicates its <code>ELF_C_NULL</code> actions and also writes any "dirty" information associated with the ELF descriptor to the file. That is, when a program has used <code>elf_getdata(3ELF)</code> or the <code>elf_flagdata(3ELF)</code> facilities to supply new (or update existing) information for an ELF descriptor, those data will be examined, coordinated, translated if necessary (see <code>elf32_xlatetof(3ELF)</code>), and written to the file. When portions of the file are written, any <code>ELF_F_DIRTY</code> bits are reset, indicating those items no longer need to be written to the file (see <code>elf_flagdata(3ELF)</code>). The sections' data are written in the order of their section header entries, and the section header table is written to the end of the file. When the ELF descriptor was created with <code>elf_begin()</code> , it must have allowed writing the file. That is, the <code>elf_begin()</code> command must have been either <code>ELF_C_RDWR</code> or <code>ELF_C_WRITE</code> .	Member	Notes
ELF_C_NULL	This value tells <code>elf_update()</code> to recalculate various values, updating only the ELF descriptor's memory structures. Any modified structures are flagged with the <code>ELF_F_DIRTY</code> bit. A program thus can update the structural information and then reexamine them without changing the file associated with the ELF descriptor. Because this does not change the file, the ELF descriptor may allow reading, writing, or both reading and writing (see <code>elf_begin(3ELF)</code>).						
ELF_C_WRITE	If <i>cmd</i> has this value, <code>elf_update()</code> duplicates its <code>ELF_C_NULL</code> actions and also writes any "dirty" information associated with the ELF descriptor to the file. That is, when a program has used <code>elf_getdata(3ELF)</code> or the <code>elf_flagdata(3ELF)</code> facilities to supply new (or update existing) information for an ELF descriptor, those data will be examined, coordinated, translated if necessary (see <code>elf32_xlatetof(3ELF)</code>), and written to the file. When portions of the file are written, any <code>ELF_F_DIRTY</code> bits are reset, indicating those items no longer need to be written to the file (see <code>elf_flagdata(3ELF)</code>). The sections' data are written in the order of their section header entries, and the section header table is written to the end of the file. When the ELF descriptor was created with <code>elf_begin()</code> , it must have allowed writing the file. That is, the <code>elf_begin()</code> command must have been either <code>ELF_C_RDWR</code> or <code>ELF_C_WRITE</code> .						
Member	Notes						

e_ident[EI_DATA]	Library controls other e_ident values
e_type	
e_machine	
e_version	
e_entry	
e_phoff	Only when ELF_F_LAYOUT asserted
e_shoff	Only when ELF_F_LAYOUT asserted
e_flags	
e_shstrndx	

The following table shows the Program Header members:

Member	Notes
p_type	The application controls all program header entries
p_offset	
p_vaddr	
p_paddr	
p_filesz	
p_memsz	
p_flags	
p_align	

The following table shows the Section Header members:

Member	Notes
sh_name	
sh_type	
sh_flags	

elf_update(3ELF)

sh_addr	
sh_offset	Only when <code>ELF_F_LAYOUT</code> asserted
sh_size	Only when <code>ELF_F_LAYOUT</code> asserted
sh_link	
sh_info	
sh_addralign	Only when <code>ELF_F_LAYOUT</code> asserted
sh_entsize	

The following table shows the Data Descriptor members:

Member	Notes
d_buf	
d_type	
d_size	
d_off	Only when <code>ELF_F_LAYOUT</code> asserted
d_align	
d_version	

Note that the program is responsible for two particularly important members (among others) in the ELF header. The `e_version` member controls the version of data structures written to the file. If the version is `EV_NONE`, the library uses its own internal version. The `e_ident[EI_DATA]` entry controls the data encoding used in the file. As a special case, the value may be `ELFDATANONE` to request the native data encoding for the host machine. An error occurs in this case if the native encoding doesn't match a file encoding known by the library.

Further note that the program is responsible for the `sh_entsize` section header member. Although the library sets it for sections with known types, it cannot reliably know the correct value for all sections. Consequently, the library relies on the program to provide the values for unknown section types. If the entry size is unknown or not applicable, the value should be set to 0.

When deciding how to build the output file, `elf_update()` obeys the alignments of individual data buffers to create output sections. A section's most strictly aligned data buffer controls the section's alignment. The library also inserts padding between buffers, as necessary, to ensure the proper alignment of each buffer.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO elf(3ELF), elf32_fsize(3ELF), elf32_getehdr(3ELF), elf32_getshdr(3ELF), elf32_xlatetof(3ELF), elf_begin(3ELF), elf_flagdata(3ELF), elf_getdata(3ELF), libelf(3LIB), attributes(5)

NOTES As mentioned above, the ELF_C_WRITE command translates data as necessary, before writing them to the file. This translation is *not* always transparent to the application program. If a program has obtained pointers to data associated with a file (for example, see elf32_getehdr(3ELF) and elf_getdata(3ELF)), the program should reestablish the pointers after calling elf_update().

elf_version(3ELF)

NAME	elf_version – coordinate ELF library and application versions						
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> unsigned elf_version(unsigned <i>ver</i>);</pre>						
DESCRIPTION	As elf(3ELF) explains, the program, the library, and an object file have independent notions of the latest ELF version. <code>elf_version()</code> lets a program query the ELF library's <i>internal version</i>. It further lets the program specify what memory types it uses by giving its own <i>working version</i>, <code>ver</code>, to the library. Every program that uses the ELF library must coordinate versions as described below. The header <code><libelf.h></code> supplies the version to the program with the macro <code>EV_CURRENT</code>. If the library's internal version (the highest version known to the library) is lower than that known by the program itself, the library may lack semantic knowledge assumed by the program. Accordingly, <code>elf_version()</code> will not accept a working version unknown to the library. Passing <code>ver</code> equal to <code>EV_NONE</code> causes <code>elf_version()</code> to return the library's internal version, without altering the working version. If <code>ver</code> is a version known to the library, <code>elf_version()</code> returns the previous (or initial) working version number. Otherwise, the working version remains unchanged and <code>elf_version()</code> returns <code>EV_NONE</code>.						
EXAMPLES	EXAMPLE 1 A sample display of using the <code>elf_version()</code> function. The following excerpt from an application program protects itself from using an older library: <pre>if (elf_version(EV_CURRENT) == EV_NONE) { /* library out of date */ /* recover from error */ }</pre>						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe						
SEE ALSO	<code>elf(3ELF)</code> , <code>elf32_xlatetof(3ELF)</code> , <code>elf_begin(3ELF)</code> , <code>libelf(3LIB)</code> , <code>attributes(5)</code>						
NOTES	The working version should be the same for all operations on a particular ELF descriptor. Changing the version between operations on a descriptor will probably not give the expected results.						

NAME	erf, erfc – error and complementary error functions				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double erf(double <i>x</i>) ; double erfc(double <i>x</i>) ;</pre>				
DESCRIPTION	The <code>erf()</code> function computes the error function of x, defined as: $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ The <code>erfc()</code> function computes $1.0 - \text{erf}(x)$.				
RETURN VALUES	Upon successful completion, <code>erf()</code> and <code>erfc()</code> return the value of the error function and complementary error function, respectively. If x is NaN, NaN is returned.				
ERRORS	No errors will occur.				
USAGE	The <code>erfc()</code> function is provided because of the extreme loss of relative accuracy if <code>erf(x)</code> is called for large x and the result subtracted from 1.0.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>attributes(5)</code>				

exp(3M)

NAME	exp – exponential function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double exp(double <i>x</i>);</pre>				
DESCRIPTION	The <code>exp()</code> function computes the exponential of x , defined as e^x .				
RETURN VALUES	Upon successful completion, <code>exp()</code> returns the exponential of x. If the correct value would cause overflow, <code>exp()</code> returns <code>HUGE_VAL</code> and sets <code>errno</code> to <code>ERANGE</code>. If the correct value would cause underflow to zero, <code>exp()</code> returns 0 and may set <code>errno</code> to <code>ERANGE</code>. If x is NaN, NaN is returned. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>exp()</code> function will fail if: <code>ERANGE</code> The result overflows. The <code>exp()</code> function may fail if: <code>ERANGE</code> The result underflows.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>exp()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>log(3M)</code> , <code>matherr(3M)</code> , <code>mp(3MP)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				
NOTES	Prior to Solaris 2.6, there was a conflict between the <code>pow</code> function in this library and the <code>pow</code> function in the <code>libmp</code> library. This conflict was resolved by prepending <code>mp_</code> to all functions in the <code>libmp</code> library. See <code>mp(3MP)</code> for details.				

NAME	expm1 – computes exponential functions				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double expm1 (double <i>x</i>) ;</pre>				
DESCRIPTION	The expm1 () function computes $e^x - 1.0$.				
RETURN VALUES	If x is NaN, then the function returns NaN. If x is positive infinity, expm1 () returns positive infinity. If x is negative infinity, expm1 () returns -1.0. If the value overflows, expm1 () returns HUGE_VAL.				
ERRORS	No errors will occur.				
USAGE	The value of expm1 (x) may be more accurate than $\exp (x) - 1.0$ for small values of x. The expm1 () and log1p(3M) functions are useful for financial calculations of $((1+x)^n - 1)/x$, namely: <pre>expm1 (<i>n</i> * log1p (<i>x</i>)) / <i>x</i></pre> when x is very small (for example, when performing calculations with a small daily interest rate). These functions also simplify writing accurate inverse hyperbolic functions.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	exp(3M), ilogb(3M), log1p(3M), attributes(5)				

fabs(3M)

NAME	fabs – absolute value function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double fabs(double <i>x</i>);</pre>				
DESCRIPTION	The <code>fabs()</code> function computes the absolute value of x , $ x $.				
RETURN VALUES	Upon successful completion, <code>fabs()</code> returns the absolute value of x . If x is NaN, NaN is returned.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>attributes(5)</code>				

NAME	floor – floor function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double floor(double <i>x</i>);</pre>				
DESCRIPTION	The <code>floor()</code> function computes the largest integral value not greater than <i>x</i> .				
RETURN VALUES	Upon successful completion, <code>floor()</code> returns the largest integral value not greater than <i>x</i>, expressed as a double. If <i>x</i> is NaN, NaN is returned. If <i>x</i> is $\pm\text{Inf}$ or ± 0, <i>x</i> is returned.				
ERRORS	No errors will occur.				
USAGE	The integral value returned by <code>floor()</code> as a double might not be expressible as an <code>int</code> or <code>long int</code> . The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>ceil(3M)</code> , <code>isnan(3M)</code> , <code>attributes(5)</code>				

fmod(3M)

NAME	fmod – floating-point remainder value function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double fmod(double <i>x</i>, double <i>y</i>);</pre>				
DESCRIPTION	The <code>fmod()</code> function returns the floating-point remainder of the division of <i>x</i> by <i>y</i> .				
RETURN VALUES	The <code>fmod()</code> function returns the value $x - i * y$, for some integer <i>i</i> such that, if <i>y</i> is non-zero, the result has the same sign as <i>x</i> and magnitude less than the magnitude of <i>y</i>. If <i>x</i> or <i>y</i> is NaN, NaN is returned. If <i>y</i> is 0, NaN is returned and <code>errno</code> is set to <code>EDOM</code>. If <i>x</i> is $\pm\text{Inf}$, NaN is returned. If <i>y</i> is non-zero, <code>fmod(± 0, <i>y</i>)</code> returns the value of <i>x</i>. If <i>x</i> is not $\pm\text{Inf}$, <code>fmod(<i>x</i>, $\pm\text{Inf}$)</code> returns the value of <i>x</i>.				
ERRORS	The <code>fmod()</code> function may fail if: <code>EDOM</code> <i>y</i> is 0. No other errors will occur.				
USAGE	Portable applications should not call <code>fmod()</code> with <i>y</i> equal to 0, because the result is implementation-dependent. The application should verify <i>y</i> is non-zero before calling <code>fmod()</code>. An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>fmod()</code>. If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>attributes(5)</code>				

NAME freeDmiString – free dynamic memory allocated for input DmiString structure

SYNOPSIS

```
cc [ flag ... ] file ... -ldmi -lnsl -lrwtool [ library ... ]
#include <dmi/util.hh>

void freeDmiString(DmiString_t *dstr);
```

DESCRIPTION The freeDmiString() function frees dynamic memory allocated for the input DmiString structure.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	MT-Safe

SEE ALSO newDmiString(3DMI), libdmi(3LIB), attributes(5)

gelf(3ELF)

NAME	gelf, gelf_checksum, gelf_fsize, gelf_getclass, gelf_getdyn, gelf_getehdr, gelf_getphdr, gelf_getrel, gelf_getrela, gelf_getshdr, gelf_getsym, gelf_getsyminfo, gelf_newehdr, gelf_newphdr, gelf_update_dyn, gelf_update_ehdr, gelf_update_phdr, gelf_update_rel, gelf_update_rela, gelf_update_shdr, gelf_update_sym, gelf_update_syminfo, gelf_xlatetof, gelf_xlatetom – generic class-independent ELF interface
SYNOPSIS	<pre>cc [flag ...] file ... -lelf [library ...] #include <gelf.h> long gelf_checksum(Elf *elf); int gelf_getclass(Elf *elf); size_t gelf_fsize(Elf *elf, Elf_Type type, size_t cnt, unsigned ver); GElf_Ehdr *gelf_getehdr(Elf *elf, GElf_Ehdr *dst); int gelf_update_ehdr(Elf *elf, GElf_Ehdr *src); unsigned long gelf_newehdr(Elf *elf, int class); GElf_Phdr *gelf_getphdr(Elf *elf, int ndx, GElf_Phdr *dst); int gelf_update_phdr(Elf *elf, int ndx, GElf_Phdr *src); unsigned long gelf_newphdr(Elf *elf, size_t phnum); GElf_Shdr *gelf_getshdr(Elf_Scn *scn, GElf_Shdr *dst); int gelf_update_shdr(Elf_Scn *scn, GElf_Shdr *src); Elf_Data *gelf_xlatetof(Elf *elf, Elf_Data *dst, const Elf_Data *src, unsigned encode); Elf_Data *gelf_xlatetom(Elf *elf, Elf_Data *dst, const Elf_Data *src, unsigned encode); GElf_Sym *gelf_getsym(Elf_Data *data, int ndx, GElf_Sym *dst); int gelf_update_sym(Elf_Data *dest, int ndx, GElf_Sym *src); GElf_Dyn *gelf_getdyn(Elf_Data *src, int ndx, GElf_Dyn *dst); int gelf_update_dyn(Elf_Data *src, int ndx, GElf_Dyn *src); GElf_Rela *gelf_getrela(Elf_Data *src, int ndx, GElf_Rela *dst); int gelf_update_rela(Elf_Data *dst, int ndx, GElf_Rela *src); GElf_Rel *gelf_getrel(Elf_Data *src, int ndx, GElf_Rel *dst); int gelf_update_rel(Elf_Data *dst, int ndx, GElf_Rel *src); GElf_Syminfo *gelf_getsyminfo(Elf_Data *src, int ndx, GElf_Syminfo *dst); int gelf_update_syminfo(Elf_Data *dst, int ndx, GElf_Syminfo *src); GElf_Move *gelf_getmove(Elf_Data *src, int ndx, GElf_Move *dst);</pre>

`gelf(3ELF)`

```
int gelf_update_move(Elf_Data *dst, int ndx, GElf_Move *src);
```

DESCRIPTION

GElf is a generic, ELF class-independent API, for manipulating ELF object files. GElf provides a single, common interface for handling 32-bit and 64-bit ELF format object files. GElf is a translation layer between the application and the class-dependent parts of the ELF library. Thus, the application can use GElf, which in turn, will call the corresponding `elf32_` or `elf64_` functions on behalf of the application. The data structures returned are all large enough to hold 32-bit and 64-bit data.

GElf provides a simple, class-independent layer of indirection over the class-dependent ELF32 and ELF64 APIs. GElf is stateless, and may be used alongside the ELF32 and ELF64 APIs.

GElf always returns a copy of the underlying ELF32 or ELF64 structure, and therefore the programming practice of using the address of an ELF header as the base offset for the ELF's mapping into memory should be avoided. Also, data accessed by type-casting the `Elf_Data` buffer to a class-dependent type and treating it like an array, for example, a symbol table, will not work under GElf, and the `gelf_get` functions must be used instead. See the EXAMPLE section.

Programs which create or modify ELF files using `libelf(3LIB)` need to perform an extra step when using GElf. Modifications to GElf values must be explicitly flushed to the underlying ELF32 or ELF64 structures by way of the `gelf_update_` interfaces. Use of `elf_update` or `elf_flagelf` and the like remains the same.

The sizes of versioning structures remains the same between ELF32 and ELF64. The GElf API only defines types for versioning, rather than a functional API. The processing of versioning information will stay the same in the GElf environment as it was in the class-dependent ELF environment.

List of Functions

<code>gelf_checksum()</code>	An analog to <code>elf32_checksum(3ELF)</code> and <code>elf64_checksum(3ELF)</code> .
<code>gelf_getclass()</code>	Returns one of the constants <code>ELFCLASS32</code> , <code>ELFCLASS64</code> or <code>ELFCLASSNONE</code> .
<code>gelf_fsize()</code>	An analog to <code>elf32_fsize(3ELF)</code> and <code>elf64_fsize(3ELF)</code> .
<code>gelf_getehdr()</code>	An analog to <code>elf32_getehdr(3ELF)</code> and <code>elf64_getehdr(3ELF)</code> . <code>dst</code> points to the location where the <code>GElf_Ehdr</code> header will be stored.
<code>gelf_update_ehdr()</code>	Copies the contents of the <code>GElf_Ehdr</code> ELF header to the underlying <code>Elf32_Ehdr</code> or <code>Elf64_Ehdr</code> structure.
<code>gelf_newehdr()</code>	An analog to <code>elf32_newehdr(3ELF)</code> and <code>elf64_newehdr(3ELF)</code> .

`gelf(3ELF)`

<code>gelf_getphdr()</code>	An analog to <code>toelf32_getphdr(3ELF)</code> and <code>elf64_getphdr(3ELF)</code> . <code>dst</code> points to the location where the <code>GElf_Phdr</code> program header will be stored.
<code>gelf_update_phdr()</code>	Copies the contents of <code>GElf_Phdr</code> program header to underlying the <code>Elf32_Phdr</code> or <code>Elf64_Phdr</code> structure.
<code>gelf_newphdr()</code>	An analog to <code>elf32_newphdr(3ELF)</code> and <code>elf64_newphdr(3ELF)</code> .
<code>gelf_getshdr()</code>	An analog to <code>elf32_getshdr(3ELF)</code> and <code>elf64_getshdr(3ELF)</code> . <code>dst</code> points to the location where the <code>GElf_Shdr</code> section header will be stored.
<code>gelf_update_shdr()</code>	Copies the contents of <code>GElf_Shdr</code> section header to underlying the <code>Elf32_Shdr</code> or <code>Elf64_Shdr</code> structure.
<code>gelf_xlatetof()</code>	An analog to <code>elf32_xlatetof(3ELF)</code> and <code>elf64_xlatetof(3ELF)</code>
<code>gelf_xlatetom()</code>	An analog to <code>elf32_xlatetom(3ELF)</code> and <code>elf64_xlatetom(3ELF)</code>
<code>gelf_getsym()</code>	Retrieves the <code>Elf32_Sym</code> or <code>Elf64_Sym</code> information from the symbol table at the given index. <code>dst</code> points to the location where the <code>GElf_Sym</code> symbol entry will be stored.
<code>gelf_update_sym()</code>	Copies the <code>GElf_Sym</code> information back into the underlying <code>Elf32_Sym</code> or <code>Elf64_Sym</code> structure at the given index.
<code>gelf_getdyn()</code>	Retrieves the <code>Elf32_Dyn</code> or <code>Elf64_Dyn</code> information from the dynamic table at the given index. <code>dst</code> points to the location where the <code>GElf_Dyn</code> dynamic entry will be stored.
<code>gelf_update_dyn()</code>	Copies the <code>GElf_Dyn</code> information back into the underlying <code>Elf32_Dyn</code> or <code>Elf64_Dyn</code> structure at the given index.
<code>gelf_getrela()</code>	Retrieves the <code>Elf32_Rela</code> or <code>Elf64_Rela</code> information from the relocation table at the given index. <code>dst</code> points to the location where the <code>GElf_Rela</code> relocation entry will be stored.
<code>gelf_update_rela()</code>	Copies the <code>GElf_Rela</code> information back into the underlying <code>Elf32_Rela</code> or <code>Elf64_Rela</code> structure at the given index.

`gelf(3ELF)`

<code>gelf_getrel()</code>	Retrieves the <code>Elf32_Rel</code> or <code>Elf64_Rel</code> information from the relocation table at the given index. <code>dst</code> points to the location where the <code>GElf_Rel</code> relocation entry will be stored.
<code>gelf_update_rel()</code>	Copies the <code>GElf_Rel</code> information back into the underlying <code>Elf32_Rel</code> or <code>Elf64_Rel</code> structure at the given index.
<code>gelf_getsyminfo()</code>	Retrieves the <code>Elf32_Syminfo</code> or <code>Elf64_Syminfo</code> information from the relocation table at the given index. <code>dst</code> points to the location where the <code>GElf_Syminfo</code> symbol information entry will be stored.
<code>gelf_update_syminfo()</code>	Copies the <code>GElf_Syminfo</code> information back into the underlying <code>Elf32_Syminfo</code> or <code>Elf64_Syminfo</code> structure at the given index.
<code>gelf_getmove()</code>	Retrieves the <code>Elf32_Move</code> or <code>Elf64_Move</code> information from the move table at the given index. <code>dst</code> points to the location where the <code>GElf_Move</code> move entry will be stored.
<code>gelf_update_move()</code>	Copies the <code>GElf_Move</code> information back into the underlying <code>Elf32_Move</code> or <code>Elf64_Move</code> structure at the given index.

RETURN VALUES Upon failure, all `GElf` functions return 0 and set `elf_errno`. See `elf_errno(3ELF)`

EXAMPLES **EXAMPLE 1** Printing the ELF Symbol Table

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <libelf.h>
#include <gelf.h>

void
main(int argc, char **argv)
{
    Elf          *elf;
    Elf_Scn      *scn = NULL;
    GElf_Shdr    shdr;
    Elf_Data     *data;
    int          fd, ii, count;

    elf_version(EV_CURRENT);

    fd = open(argv[1], O_RDONLY);
    elf = elf_begin(fd, ELF_C_READ, NULL);

    while ((scn = elf_nextscn(elf, scn)) != NULL) {
```

`gelf(3ELF)`

EXAMPLE 1 Printing the ELF Symbol Table *(Continued)*

```
    gelf_getshdr(scn, &shdr);
    if (shdr.sh_type == SHT_SYMTAB) {
        /* found a symbol table, go print it. */
        break;
    }
}

data = elf_getdata(scn, NULL);
count = shdr.sh_size / shdr.sh_entsize;

/* print the symbol names */
for (ii = 0; ii < count; ++ii) {
    GElf_Sym sym;
    gelf_getsym(data, ii, &sym);
    printf("%s\n", elf_strptr(elf, shdr.sh_link, sym.st_name));
}
elf_end(elf);
close(fd);
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	Safe

SEE ALSO

`elf(3ELF)`, `elf32_checksum(3ELF)`, `elf32_fsize(3ELF)`, `elf32_getehdr(3ELF)`,
`elf32_getphdr(3ELF)`, `elf32_getshdr(3ELF)`, `elf32_newehdr(3ELF)`,
`elf32_newphdr(3ELF)`, `elf32_xlatetof(3ELF)`, `elf32_xlatetom(3ELF)`,
`elf_errno(3ELF)`, `libelf(3LIB)`, `attributes(5)`

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – get audit control file information		
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <bsm/libbsm.h> int getacdir(char *dir, int len); int getacmin(int *min_val); int getacflg(char *auditstring, int len); int getacna(char *auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry. <code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked. <code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. <code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events. Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches. Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>contains default parameters read by the audit daemon, <code>auditd(1M)</code></td></tr> </table>	<code>/etc/security/audit_control</code>	contains default parameters read by the audit daemon, <code>auditd(1M)</code>
<code>/etc/security/audit_control</code>	contains default parameters read by the audit daemon, <code>auditd(1M)</code>		
RETURN VALUES	<code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return: 0 on success. -2 on failure and set <code>errno</code> to indicate the error. <code>getacmin()</code> and <code>getacflg()</code> return: 1 on EOF. <code>getacdir()</code> returns:		

getacinfo(3BSM)

- 1 on EOF.
- 2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()`.

These functions return:

- 3 if the directory entry format in the `audit_control` file is incorrect.

`getacdir()`, `getacflg()` and `getacna()` return:

- 3 if the input buffer is too short to accommodate the record.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SEE ALSO `audit_warn(1M)`, `bsmconv(1M)`, `inetd(1M)`, `audit_control(4)`, `attributes(5)`

NOTES The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char *name); struct au_class_ent *getauclassnam_r(au_class_ent_t *class_int, const char *name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t *class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	getauclassent() and getauclassnam() each return an audit_class entry. getauclassnam() searches for an audit_class entry with a given class name <i>name</i>. getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL. setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent(). endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth. getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t, which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure. The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members: <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>
RETURN VALUES	getauclassnam() and getauclassnam_r() return a pointer to a struct au_class_ent if they successfully locate the requested entry; otherwise they return NULL.

getauclassent(3BSM)

getauclassent() and getauclassent_r() return a pointer to a struct au_class_ent if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

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

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except getauclassent() and getauclassnam. The two functions, getauclassent_r() and getauclassnam_r() have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

SEE ALSO bsmconv(1M), audit_class(4), audit_event(4), attributes(5)

NOTES All information is contained in a static area, so it must be copied if it is to be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.

NAME	getauditflags, getauditflagsbin, getauditflagschar – convert audit flag specifications				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char *<i>auditstring</i>, au_mask_t *<i>masks</i>); int getauditflagschar(char *<i>auditstring</i>, au_mask_t *<i>masks</i>, int <i>verbose</i>);</pre>				
DESCRIPTION	getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into au_mask_t fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in audit_control(4). getauditflagschar() converts the au_mask_t fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the ASCII representation of the events. <i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The au_mask_t fields pointed to by <i>masks</i> correspond to binary values defined in <bsm/audit.h>, which is read by <bsm/libbsm.h>.				
RETURN VALUES	getauditflagsbin() and getauditflagschar(): -1 is returned on error and 0 on success.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe.</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SEE ALSO	bsmconv(1M), audit.log(4), audit_control(4), attributes(5)				
BUGS	This is not a very extensible interface.				
NOTES	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.				

getauevent(3BSM)

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – get audit_event entry
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure. getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL. getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name. getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number. getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number. setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r(). endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.

getauevent(3BSM)

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

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

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

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user-ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SEE ALSO

`bsmconv(1M)`, `getauclassent(3BSM)`, `getpwnam(3C)`, `audit_class(4)`, `audit_event(4)`, `passwd(4)`, `attributes(5)`

NOTES

All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, so it must be copied if it is to be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

getauthattr(3SECDB)

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

getauthattr(3SECDB)

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

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

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

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

	/etc/security/policy.conf or	Is user
Authorization name	user_attr or prof_attr entry	authorized?
com.sun.printer.postscript	com.sun.printer.postscript	Yes
com.sun.printer.postscript	com.sun.printer.*	Yes
com.sun.printer.grant	com.sun.printer.*	No

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

RETURN VALUES

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

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

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

USAGE

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

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the _r suffix naming convention.

Individual attributes in the attr structure can be referred to by calling the kva_match(3SECDB) function.

getauthattr(3SECDB)

WARNINGS Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

FILES

/etc/nsswitch.conf	configuration file lookup information for the name server switch
/etc/user_attr	extended user attributes
/etc/security/auth_attr	authorization attributes
/etc/security/policy.conf	policy definitions
/etc/security/prof_attr	profile information

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB), auth_attr(4), nsswitch.conf(4), prof_attr(4), user_attr(4), attributes(5), rbac(5)

NAME	getauusernam, getauuserent, setauuser, endauuser – get audit_user entry
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent *getauusernam(const char *name); struct au_user_ent *getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent *getauusernam_r(au_user_ent_t *u, const char *name); struct au_user_ent *getauuserent_r(au_user_ent_t *u);</pre>
DESCRIPTION	The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry. Entries can come from any of the sources specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>). The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>. The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>. The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>. The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth. The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls. The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members: <pre>char *au_name; au_mask_t au_always; au_mask_t au_never;</pre>
RETURN VALUES	The <code>getauusernam()</code> function returns a pointer to a <code>struct au_user_ent</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code> .

getauusernam(3BSM)

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

USAGE The functionality described in this manual page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

FILES

<code>/etc/security/audit_user</code>	stores per-user audit event mask
<code>/etc/passwd</code>	stores user-id to username mappings
<code>/etc/security/audit_user</code>	stores per-user audit event mask

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

SEE ALSO `bsmconv(1M)`, `getpwnam(3C)`, `audit_user(4)`, `nsswitch.conf(4)`, `passwd(4)`, `attributes(5)`

NOTES All information for the `getauuserent()` and `getauusernam()` functions is contained in a static area, so it must be copied if it is to be saved.

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	getddent, getddnam, setddent, endddent, setddfile – get device_deallocate entry
SYNOPSIS	<pre>cc [flag...] file... -lbsm [library...] #include <bsm/devices.h> devdealloc_t *getddent(void); devdealloc_t *getddnam(char *name); void setddent(void); void endddent(void); void setddfile(char *file);</pre>
DESCRIPTION	The <code>getddent()</code> and <code>getddnam()</code> functions each return a <code>device_deallocate</code> entry. The <code>getddent()</code> function enumerates all <code>device_deallocate</code> entries. Successive calls to this function return either successive <code>device_deallocate</code> entries or NULL. The <code>getddnam()</code> function searches for a <code>device_deallocate</code> entry with a given device name. The internal representation of a <code>device_deallocate</code> entry is a <code>devdealloc_t</code> structure defined in <code><bsm/devices.h></code> with the following members: <pre>char *dd_devname; /* device allocation name */ char *dd_logout; /* deallocation action on user logout */ char *dd_boot; /* deallocation action on system boot */</pre> The <code>setddent()</code> function “rewinds” to the beginning of the enumeration of <code>device_deallocate</code> entries. Calls to <code>getddnam()</code> may leave the enumeration in an indeterminate state, so <code>setddent()</code> should be called before the first call to <code>getddent()</code>. The <code>endddent()</code> function can be called to indicate that <code>device_deallocate</code> processing is complete. The library can then close any <code>opendevice_deallocate</code> file, deallocate any internal storage, and so forth. The <code>setddfile()</code> function changes the pathname used by the other functions for opening the <code>device_deallocate</code> file, allowing use of <code>device_deallocate</code> files other than the default file, <code>/etc/security/device_deallocate</code>.
RETURN VALUES	The <code>getddent()</code> function returns a pointer to a <code>devdealloc_t</code> if it successfully enumerates an entry. Otherwise it returns NULL, indicating the end of the enumeration. The <code>getddnam()</code> function returns a pointer to a <code>devdealloc_t</code> if it successfully locates the requested entry. Otherwise it returns NULL.
FILES	<pre>/etc/security/device_deallocate</pre> Administrative file defining parameters for device deallocation.

getddent(3BSM)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO free(3C), attributes(5)

NOTES The `getddent()` and `getddnam()` functions allocate memory for the pointers they return. This memory can be deallocated with the `free(3C)` function.

NAME	getdmapent, getdmapnam, getdmaptype, getdmapdev, setdmapent, enddmapent, setdmapfile – get device_maps entry
SYNOPSIS	<pre>cc [flag...] file... -lbsm [library...] #include <bsm/devices.h> devmap_t *getdmapent(void); devmap_t *getdmapnam(char *name); devmap_t *getdmapdev(char *name); devmap_t *getdmaptype(char *type); void setdmapent(void); void enddmapent(void); void setdmapfile(char *file);</pre>
DESCRIPTION	The getdmapent(), getdmapnam(), getdmapdev(), and getdmaptype() functions each return a device_maps entry. The getdmapent() function enumerates all device_maps entries. The getdmaptype() function enumerates device_maps entries with a given device type. Successive calls to these functions return either successive device_maps entries or NULL. The getdmapnam() function searches for a device_maps entry with a given device allocation name. The getdmapdev() function searches for a device_maps entry containing a given device special file. The internal representation of a device_maps entry is a devmap_t structure defined in <bsm/devices.h> with the following members: <pre>char *dmap_devname; /* device allocation name */ char *dmap_devtype; /* generic device type */ char *dmap_devlist; /* list of associated device special files */</pre> The setdmapent() function “rewinds” to the beginning of the enumeration of device_maps entries. Calls to getdmapnam() may leave the enumeration in an indeterminate state, so setdmapent() should be called before the first call to getdmapent() or getdmaptype(). The enddmapent() function can be called to indicate that device_maps processing is complete. The library can then close any open device_maps file, deallocate any internal storage, and so forth. The setdmapfile() function changes the pathname used by the other functions for opening the device_maps file, allowing use of device_maps files other than the default file, /etc/security/device_maps.
RETURN VALUES	The getdmapent() and getdmaptype() functions return a pointer to a devmap_t if they successfully enumerate an entry. Otherwise they return NULL, indicating the end of the enumeration.

getdmapent(3BSM)

The `getdmapnam()` function returns a pointer to a `devmap_t` if it successfully locates the requested entry. Otherwise it returns `NULL`.

FILES `/etc/security/device_maps`
Administrative file defining the mapping of device special files to allocatable device names.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `allocate(1)`, `free(3C)`, `device_maps(4)`, `attributes(5)`

NOTES The `getdmapent()`, `getdmapnam()`, `getdmapdev()`, and `getdmaptype()` functions allocate memory for the pointers they return. This memory can be deallocated with the `free(3C)` function.

NAME	getexecattr, free_execattr, setexecattr, endexecattr, getexecuser, getexecprof, match_execattr – get execution profile entry
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsecdb -lsocket -lnsl -lintl [<i>library...</i>] #include <exec_attr.h> #include <secdb.h> execattr_t *getexecattr(void); void free_execattr(execattr_t *ep); void setexecattr(void); void endexecattr(void); execattr_t *getexecuser(const char *username, const char *type, const char *id, int search_flag); execattr_t *getexecprof(const char *profname, const char *type, const char *id, int search_flag); execattr_t *match_execattr(execattr_t *ep, char *profname, char *type, char *id);</pre>
DESCRIPTION	The <code>getexecattr()</code> function returns a single <code>exec_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. Successive calls to <code>getexecattr()</code> return either successive <code>exec_attr</code> entries or NULL. Because <code>getexecattr()</code> always returns a single entry, the next pointer in the <code>execattr_t</code> data structure points to NULL. The internal representation of an <code>exec_attr</code> entry is an <code>execattr_t</code> structure defined in <code><exec_attr.h></code> with the following members: <pre>char name; /* name of the profile */ char type; /* type of profile */ char policy; /* policy under which the attributes are */ /* relevant*/ char res1; /* reserved for future use */ char res2; /* reserved for future use */ char id; /* unique identifier */ kva_t attr; /* attributes */ struct execattr_s next; /* optional pointer to next profile */</pre> The <code>free_execattr()</code> function releases memory. It follows the next pointers in the <code>execattr_t</code> structure so that the entire linked list is released. The <code>setexecattr()</code> function “rewinds” to the beginning of the enumeration of <code>exec_attr</code> entries. Calls to <code>getexecuser()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setexecattr()</code> should be called before the first call to <code>getexecattr()</code>. The <code>endexecattr()</code> function can be called to indicate that <code>exec_attr</code> processing is complete; the library can then close any open <code>exec_attr</code> file, deallocate any internal storage, and so forth.

getexecattr(3SECDB)

The `getexecuser()` function returns a linked list of entries that match the *type* and *id* arguments and have a profile that has been assigned to the user specified by *username*, as described in `passwd(4)`. Profiles for the user are obtained from the list of default profiles in `/etc/security/policy.conf` (see `policy.conf(4)`) and the `user_attr(4)` database. Only entries in the name service scope for which the corresponding profile entry is found in the `prof_attr(4)` database are returned.

The `getexecprof()` function returns a linked list of entries that match the *type* and *id* arguments and have the profile specified by the *profname* argument. Only entries in the name service scope for which the corresponding profile entry is found in the `prof_attr` database are returned.

Using `getexecuser()` and `getexecprof()`, programmers can search for any *type* argument, such as the manifest constant `KV_COMMAND`. The arguments are logically AND-ed together so that only entries exactly matching all of the arguments are returned. Wildcard matching applies if there is no exact match for an ID. Any argument can be assigned the `NULL` value to indicate that it is not used as part of the matching criteria. The `search_flag` controls whether the function returns the first match (`GET_ONE`), setting the next pointer to `NULL` or all matching entries (`GET_ALL`), using the next pointer to create a linked list of all entries that meet the search criteria. See `EXAMPLES`.

Once a list of entries is returned by `getexecuser()` or `getexecprof()`, the convenience function `match_execattr()` can be used to identify an individual entry. It returns a pointer to the individual element with the same profile name (*profname*), type name (*type*), and *id*. Function parameters set to `NULL` are not used as part of the matching criteria. In the event that multiple entries meet the matching criteria, only a pointer to the first entry is returned. The `kva_match(3SECDB)` function can be used to look up a key in a key-value array.

RETURN VALUES

Those functions returning data only return data related to the active policy. The `getexecattr()` function returns a pointer to a `execattr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

USAGE

The `getexecattr()`, `getexecuser()`, and `getexecprof()` functions all allocate memory for the pointers they return. This memory should be deallocated with the `free_execattr()` call. The `match_execattr()` function does not allocate any memory. Therefore, pointers returned by this function should not be deallocated.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

EXAMPLES

EXAMPLE 1 The following finds all profiles that have the `ping` command.

```
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping",
GET_ONE)) == NULL) {
```

EXAMPLE 1 The following finds all profiles that have the ping command. (Continued)

```
    /* do error */
}
```

EXAMPLE 2 The following finds the entry for the ping command in the Network Administration Profile.

```
if ((execprof=getexecprof("Network Administration", KV_COMMAND,
    "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}
```

EXAMPLE 3 The following tells everything that can be done in the Filesystem Security profile.

```
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL,
    GET_ALL))==NULL) {
    /* do error */
}
```

EXAMPLE 4 The following tells if the tar command is in a profile assigned to user wetmore. If there is no exact profile entry, the wildcard (*), if defined, is returned.

```
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar",
    GET_ONE))==NULL) {
    /* do error */
}
```

FILES	/etc/nsswitch.conf	configuration file lookup information for the name server switch
	/etc/user_attr	extended user attributes
	/etc/security/exec_attr	execution profiles
	/etc/security/policy.conf	policy definitions

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO getauthattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB), exec_attr(4), passwd(4), policy.conf(4), prof_attr(4), user_attr(4), attributes(5)

getfauditflags(3BSM)

NAME	getfauditflags – generates the process audit state				
SYNOPSIS	<pre>cc [flag ...] file ... -lbsm -lsocket -lnsl -lintl [library ...] #include <sys/param.h> #include <bsm/libbsm.h> int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);</pre>				
DESCRIPTION	getfauditflags() generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the audit_control(4) file. getfauditflags() obtains the system audit value by calling getacflg() (see getacinfo(3BSM)). usremasks points to au_mask_t fields which contains two values. The first value defines which events are <i>always</i> to be audited when they succeed. The second value defines which events are always to be audited when they fail. usrdmasks also points to au_mask_t fields which contains two values. The first value defines which events are <i>never</i> to be audited when they succeed. The second value defines which events are never to be audited when they fail. The structures pointed to by usremasks and usrdmasks may be obtained from the audit_user(4) file by calling getauusernam() which returns a pointer to a strucure containing all audit_user(4) fields for a user. The output of this function is stored in lastmasks which is a pointer of type au_mask_t as well. The first value defines which events are to be audited when they succeed and the second defines which events are to be audited when they fail. Both usremasks and usrdmasks override the values in the system audit values.				
RETURN VALUES	–1 is returned on error and 0 on success.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe.</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SEE ALSO	bsmconv(1M), getacinfo(3BSM), getauditflags(3BSM), getauusernam(3BSM), audit.log(4), audit_control(4), audit_user(4), attributes(5)				
NOTES	The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See bsmconv(1M) for more information.				

NAME	getprofattr, getprofnam, free_profattr, setprofattr, endprofattr, getproflist, free_proflist – get profile description and attributes
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsecdB -lsocket -lnsl -lintl [<i>library...</i>] #include <prof.h> profattr_t *getprofattr(void); profattr_t *getprofnam(const char *name); void free_profattr(profattr_t *pd); void setprofattr(void); void endprofattr(void); void getproflist(const char *profname, char **proflist, int *profcnt); void free_proflist(char **proflist, int profcnt);</pre>
DESCRIPTION	The <code>getprofattr()</code> and <code>getprofnam()</code> functions each return a <code>prof_attr</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getprofattr()</code> function enumerates <code>prof_attr</code> entries. The <code>getprofnam()</code> function searches for a <code>prof_attr</code> entry with a given <i>name</i>. Successive calls to these functions return either successive <code>prof_attr</code> entries or <code>NULL</code>. The internal representation of a <code>prof_attr</code> entry is a <code>profattr_t</code> structure defined in <code><prof_attr.h></code> with the following members: <pre>char name; /* Name of the profile */ char res1; /* Reserved for future use */ char res2; /* Reserved for future use */ char desc; /* Description/Purpose of the profile */ kva_t attr; /* Profile attributes */</pre> The <code>free_profattr()</code> function releases memory allocated by the <code>getprofattr()</code> and <code>getprofnam()</code> functions. The <code>setprofattr()</code> function “rewinds” to the beginning of the enumeration of <code>prof_attr</code> entries. Calls to <code>getprofnam()</code> can leave the enumeration in an indeterminate state. Therefore, <code>setprofattr()</code> should be called before the first call to <code>getprofattr()</code>. The <code>endprofattr()</code> function may be called to indicate that <code>prof_attr</code> processing is complete; the system may then close any open <code>prof_attr</code> file, deallocate storage, and so forth. The <code>getproflist()</code> function searches for the list of sub-profiles found in the given <i>profname</i> and allocates memory to store this list in <i>proflist</i>. The given <i>profname</i> will be included in the list of sub-profiles. The <i>profcnt</i> argument indicates the number of items currently valid in <i>proflist</i>. Memory allocated by <code>getproflist()</code> should be freed using the <code>free_proflist()</code> function.

getprofattr(3SECDB)

	The <code>free_proflist()</code> function frees memory allocated by the <code>getproflist()</code> function. The <i>profcnt</i> argument specifies the number of items to free from the <i>proflist</i> argument.				
RETURN VALUES	The <code>getprofattr()</code> function returns a pointer to a <code>profattr_t</code> if it successfully enumerates an entry; otherwise it returns <code>NULL</code>, indicating the end of the enumeration. The <code>getprofnam()</code> function returns a pointer to a <code>profattr_t</code> if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.				
USAGE	Individual attributes in the <code>prof_attr_t</code> structure can be referred to by calling the <code>kva_match(3SECDB)</code> function. Because the list of legal keys is likely to expand, any code must be written to ignore unknown key-value pairs without error. The <code>getprofattr()</code> and <code>getprofnam()</code> functions both allocate memory for the pointers they return. This memory should be deallocated with the <code>free_profattr()</code> function. Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the <code>_r</code> suffix naming convention.				
FILES	<code>/etc/security/prof_attr</code> profiles and their descriptions				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>auths(1)</code> , <code>profiles(1)</code> , <code>getexecattr(3SECDB)</code> , <code>getauthattr(3SECDB)</code> , <code>prof_attr(4)</code>				

NAME	getproject, getprojbyname, getprojbyid, getdefaultproj, inproj, getprojidbyname, setproject, endproject, fgetproject – project database entry functions
SYNOPSIS	<pre>cc [flag...] file... -lproject [library...] #include <project.h> struct project *getproject(struct project *proj, void *buffer, size_t bufsize); struct project *getprojbyname(const char *name, struct project *proj, void *buffer, size_t bufsize); struct project *getprojbyid(projid_t projid, struct project *proj, void *buffer, size_t bufsize); struct project *getdefaultproj(const char *username, struct project *proj, void *buffer, size_t bufsize); int inproj(const char *username, const char *projname, void *buffer, size_t bufsize); projid_t getprojidbyname(const char *name); void setproject(void); void endproject(void); struct project *fgetproject(FILE *f, struct project *proj, void *buffer, size_t bufsize);</pre>
DESCRIPTION	These functions are used to obtain entries describing user projects. Entries can come from any of the sources for a project specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>). The <code>setproject()</code>, <code>getproject()</code>, and <code>endproject()</code> functions are used to enumerate project entries from the database. The <code>setproject()</code> function effectively rewinds the project database to allow repeated searches. It sets (or resets) the enumeration to the beginning of the set of project entries. This function should be called before the first call to <code>getproject()</code>. The <code>getproject()</code> function returns a pointer to a structure containing the broken-out fields of an entry in the project database. When first called, <code>getproject()</code> returns a pointer to a project structure containing the first project structure in the project database. Successive calls can be used to read the entire database. The <code>endproject()</code> function closes the project database and deallocates resources when processing is complete. It is permissible, though possibly less efficient, for the process to call more project functions after calling <code>endproject()</code>. The <code>getprojbyname()</code> function searches the project database for an entry with the project name specified by the character string <code>name</code>.

getprojent(3PROJECT)

The `getprojbyid()` function searches the project database for an entry with the (numeric) project ID specified by *projid*.

The `getdefaultproj()` function first looks up the project key word in the `user_attr` database used to define user attributes in restricted Solaris environments. If the database is available and the keyword is present, the function looks up the named project, returning `NULL` if it cannot be found or if the user is not a member of the named project. If absent, the function looks for a match in the project database for the special project `user.username`. If no match is found, or if the user is excluded from project `user.username`, the function looks at the default group entry of the `passwd` database for the user, and looks for a match in the project database for the special name `group.groupname`, where *groupname* is the default group associated with the password entry corresponding to the given *username*. If no match is found, or if the user is excluded from project `group.groupname`, the function returns `NULL`. A special project entry called 'default' can be looked up and used as a last resort, unless the user is excluded from project 'default'. On successful lookup, this function returns a pointer to the valid project structure. By convention, the user must have a default project defined on a system to be able to log on to that system.

The `inproj()` function checks if the user specified by *username* is able to use the project specified by *projname*. This function returns 1 if the user belongs to the list of project's users, if there is a project's group that contains the specified user, if project is a user's default project, or if project's user or group list contains "*" wildcard. In all other cases it returns 0.

The `getprojidbyname()` function searches the project database for an entry with the project name specified by the character string *name*. This function returns the project ID if the requested entry is found; otherwise it returns -1.

The `fgetprojent()` function, unlike the other functions described above, does not use `nsswitch.conf`; it reads and parses the next line from the stream *f*, which is assumed to have the format of the `project(4)` file. This function returns the same values as `getprojent()`.

The `getprojent()`, `getprojbyname()`, `getprojbyid()`, `getdefaultproj()`, and `inproj()` functions are reentrant interfaces for operations with the project database. These functions use buffers supplied by the caller to store returned results and are safe for use in both single-threaded and multithreaded applications.

Reentrant interfaces require the additional arguments *proj*, *buffer*, and *bufsize*. The *proj* argument must be a pointer to a `struct project` structure allocated by the caller. On successful completion, the function returns the project entry in this structure. Storage referenced by the `project` structure is allocated from the memory provided with the *buffer* argument, which is *bufsize* bytes in size.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. The `setprojent()` function can be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprojent()`, the threads will

getproject(3PROJECT)

RETURN VALUES

enumerate disjoint subsets of the project database. The `inproj()`, `getprojbyname()`, `getprojbyid()`, and `getdefaultproj()` functions leave the enumeration position in an indeterminate state.

Project entries are represented by the `struct project` structure defined in `<project.h>`.

```
struct project {
    char      *pj_name;          /* name of the project */
    projid_t  pj_projid;        /* numerical project id */
    char      *pj_comment;       /* project comment */
    char      **pj_users;        /* vector of pointers to project
                                user names */
    char      **pj_groups;       /* vector of pointers to project
                                group names */
    char      *pj_attr;          /* project attributes */
};
```

The `getprojbyname()` and `getprojbyid()` functions each return a pointer to a `struct project` if they successfully locate the requested entry; otherwise they return `NULL`.

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

The `getprojidbyname()` function returns the project ID if the requested entry is found; otherwise it returns `-1` and sets `errno` to indicate the error.

When the pointer returned by the reentrant functions `getprojbyname()`, `getprojbyid()`, and `getproject()` is non-null, it is always equal to the *proj* pointer that was supplied by the caller.

Upon failure, `NULL` is returned and `errno` is set to indicate the error.

ERRORS

The `getproject()`, `getprojbyname()`, `getprojbyid()`, `inproj()`, `getprojidbyname()`, `fgetproject()`, and `getdefaultproj()` functions will fail if:

EINTR	A signal was caught during the operation.
EIO	An I/O error has occurred.
EMFILE	There are <code>OPEN_MAX</code> file descriptors currently open in the calling process.
ENFILE	The maximum allowable number of files is currently open in the system.
ERANGE	Insufficient storage was supplied by <i>buffer</i> and <i>bufsize</i> to contain the data to be referenced by the resulting <code>project</code> structure.

getproject(3PROJECT)

These functions can also fail if the name service switch does not specify valid project(4) name service sources. In the case of an incompletely configured name service switch configuration, getprojbyid() and other functions can return error values other than those documented above. These conditions usually occur when the nsswitch.conf file indicates that one or more name services is providing entries for the project database when that name service does not actually make a project table available.

USAGE When compiling multithreaded applications, see intro(3), Notes On Multithreaded Applications.

Applications that use the interfaces described on this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at runtime.

Use of the enumeration interface getproject() is discouraged. Enumeration is supported for the project file, NIS, and LDAP but in general is not efficient. The semantics of enumeration are discussed further in nsswitch.conf(4).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	See "Reentrant Interfaces" in Description

SEE ALSO intro(3), sysconf(3C), nsswitch.conf(4), project(4), attributes(5)

NAME	getuserattr, getusernam, getuseruid, free_userattr, setuserattr, enduserattr – get user_attr entry
SYNOPSIS	<pre>cc [flag...] file... - lsecdb - lsocket - lns1 - lint1 [library...] #include <user_attr.h> userattr_t *getuserattr(void); userattr_t *getusernam(const char *name); userattr_t *getuseruid(uid_t uid); void free_userattr(userattr_t *userattr); void setuserattr(void); void enduserattr(void);</pre>
DESCRIPTION	The <code>getuserattr()</code>, <code>getusernam()</code>, and <code>getuseruid()</code> functions each return a <code>user_attr(4)</code> entry. Entries can come from any of the sources specified in the <code>nsswitch.conf(4)</code> file. The <code>getuserattr()</code> function enumerates <code>user_attr</code> entries. The <code>getusernam()</code> function searches for a <code>user_attr</code> entry with a given user name <i>name</i>. The <code>getuseruid()</code> function searches for a <code>user_attr</code> entry with a given user id <i>uid</i>. Successive calls to these functions return either successive <code>user_attr</code> entries or NULL. The <code>free_userattr()</code> function releases memory allocated by the <code>getusernam()</code> and <code>getuserattr()</code> functions. The internal representation of a <code>user_attr</code> entry is a <code>userattr_t</code> structure defined in <code><user_attr.h></code> with the following members: <pre>char name; /* name of the user */ char qualifier; /* reserved for future use */ char res1; /* reserved for future use */ char res2; /* reserved for future use */ kva_t attr; /* list of attributes */</pre> The <code>setuserattr()</code> function “rewinds” to the beginning of the enumeration of <code>user_attr</code> entries. Calls to <code>getusernam()</code> may leave the enumeration in an indeterminate state, so <code>setuserattr()</code> should be called before the first call to <code>getuserattr()</code>. The <code>enduserattr()</code> function may be called to indicate that <code>user_attr</code> processing is complete; the library may then close any open <code>user_attr</code> file, deallocate any internal storage, and so forth.
RETURN VALUES	The <code>getuserattr()</code> function returns a pointer to a <code>userattr_t</code> if it successfully enumerates an entry; otherwise it returns NULL, indicating the end of the enumeration. The <code>getusernam()</code> function returns a pointer to a <code>userattr_t</code> if it successfully locates the requested entry; otherwise it returns NULL.

getuserattr(3SECDB)

USAGE The `getuserattr()` and `getusernam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_userattr()` function.

Applications that use the interfaces described in this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time. Note that these interfaces are reentrant even though they do not use the `_r` suffix naming convention.

Individual attributes may be referenced in the `attr` structure by calling the `kva_match(3SECDB)` function.

WARNINGS Because the list of legal keys is likely to expand, code must be written to ignore unknown key-value pairs without error.

FILES `/etc/user_attr` extended user attributes
`/etc/nsswitch.conf` configuration file lookup information for the name server switch

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `getauthattr(3SECDB)`, `getexecattr(3SECDB)`, `getprofattr(3SECDB)`, `user_attr(4)`, `attributes(5)`

NAME	gmatch – shell global pattern matching				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int gmatch(const char *<i>str</i>, const char *<i>pattern</i>) ;</pre>				
DESCRIPTION	gmatch() checks whether the null-terminated string <i>str</i> matches the null-terminated pattern string <i>pattern</i> . See the sh(1), section File Name Generation, for a discussion of pattern matching. A backslash (\) is used as an escape character in pattern strings.				
RETURN VALUES	gmatch() returns non-zero if the pattern matches the string, zero if the pattern does not.				
EXAMPLES	EXAMPLE 1 Examples of gmatch() function. In the following example, gmatch() returns non-zero (true) for all strings with “a” or “-” as their last character. <pre>char *s; gmatch (s, "[a\-")</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	sh(1), attributes(5)				
NOTES	When compiling multithreaded applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multithreaded applications.				

hypot(3M)

NAME	hypot – Euclidean distance function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double hypot(double <i>x</i>, double <i>y</i>);</pre>				
DESCRIPTION	The <code>hypot()</code> function computes the length of the hypotenuse of a right-angled triangle: $\sqrt{x^2+y^2}$				
RETURN VALUES	Upon successful completion, <code>hypot()</code> returns the length of the hypotenuse of a right angled triangle with sides of length <i>x</i> and <i>y</i>. If the result would cause overflow, <code>HUGE_VAL</code> is returned and <code>errno</code> may be set to <code>ERANGE</code>. If <i>x</i> or <i>y</i> is NaN, NaN is returned.				
ERRORS	The <code>hypot()</code> function may fail if: <table><tr><td><code>ERANGE</code></td><td>The result overflows.</td></tr></table>	<code>ERANGE</code>	The result overflows.		
<code>ERANGE</code>	The result overflows.				
USAGE	The <code>hypot()</code> function takes precautions against underflow and overflow during intermediate steps of the computation. An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>hypot()</code>. If <code>errno</code> is non-zero on return, or the return value is <code>HUGE_VAL</code> or NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>sqrt(3M)</code> , <code>attributes(5)</code>				

NAME	idn_decodename, idn_decodename2, idn_enable, idn_encodename, idn_nameinit – IDN (Internationalized Domain Name) conversion functions																
SYNOPSIS	<pre>cc [flag...] file... -lidnkit [library...] #include <idn/api.h> idn_result_t idn_decodename(int actions, const char *from, char *to, size_t tolen); idn_result_t idn_decodename2(int actions, const char *from, char *to, size_t tolen, const char *auxencoding); idn_result_t idn_encodename(int actions, const char *from, char *to, size_t tolen); void idn_enable(int on_off); idn_result_t idn_nameinit(int load_file);</pre>																
DESCRIPTION	The <code>idn_nameinit()</code> function initializes the library. It also sets default configuration if <code>load_file</code> is 0, otherwise it tries to read a configuration file. If <code>idn_nameinit()</code> is called more than once, the library initialization will take place only at the first call while the actual configuration procedure will occur at every call. If there are no errors, <code>idn_nameinit()</code> returns <code>idn_success</code>. Otherwise, the returned value indicates the cause of the error. See the section RETURN VALUES below for the error codes. It is usually not necessary to call this function explicitly because it is implicitly called when <code>idn_encodename()</code>, <code>idn_decodename()</code>, or <code>idn_decodename2()</code> is first called without prior calling of <code>idn_nameinit()</code>. The <code>idn_encodename()</code> function performs name preparation and encoding conversion on the internationalized domain name specified by <i>from</i>, and stores the result to <i>to</i>, whose length is specified by <i>tolen</i>. The <i>actions</i> argument is a bitwise-OR of the following macros, specifying which subprocesses in the encoding process are to be employed. <table> <tr> <td>IDN_LOCALCONV</td><td>Local encoding to UTF-8 conversion</td></tr> <tr> <td>IDN_DELIMMAP</td><td>Delimiter mapping</td></tr> <tr> <td>IDN_LOCALMAP</td><td>Local mapping</td></tr> <tr> <td>IDN_NAMEPREP</td><td>NAMEPREP mapping, normalization, prohibited character check, and bidirectional string check</td></tr> <tr> <td>IDN_UNASCHECK</td><td>NAMEPREP unassigned codepoint check</td></tr> <tr> <td>IDN_ASCCHECK</td><td>ASCII range character check</td></tr> <tr> <td>IDN_IDNCONV</td><td>UTF-8 to IDN encoding conversion</td></tr> <tr> <td>IDN_LENCHECK</td><td>Label length check</td></tr> </table>	IDN_LOCALCONV	Local encoding to UTF-8 conversion	IDN_DELIMMAP	Delimiter mapping	IDN_LOCALMAP	Local mapping	IDN_NAMEPREP	NAMEPREP mapping, normalization, prohibited character check, and bidirectional string check	IDN_UNASCHECK	NAMEPREP unassigned codepoint check	IDN_ASCCHECK	ASCII range character check	IDN_IDNCONV	UTF-8 to IDN encoding conversion	IDN_LENCHECK	Label length check
IDN_LOCALCONV	Local encoding to UTF-8 conversion																
IDN_DELIMMAP	Delimiter mapping																
IDN_LOCALMAP	Local mapping																
IDN_NAMEPREP	NAMEPREP mapping, normalization, prohibited character check, and bidirectional string check																
IDN_UNASCHECK	NAMEPREP unassigned codepoint check																
IDN_ASCCHECK	ASCII range character check																
IDN_IDNCONV	UTF-8 to IDN encoding conversion																
IDN_LENCHECK	Label length check																

idn_decodename(3EXT)

Details of this encoding process can be found in the section Name Encoding

For convenience, also `IDN_ENCODE_QUERY`, `IDN_ENCODE_APP`, and `IDN_ENCODE_STORED` macros are provided. `IDN_ENCODE_QUERY` is used to encode a “query string” (see the IDNA specification). It is equal to:

```
(IDN_LOCALCONV | IDN_DELIMMAP | IDN_LOCALMAP | IDN_NAMEPREP |  
IDN_IDNCONV | IDN_LENCHECK)
```

`IDN_ENCODE_APP` is used for ordinary application to encode a domain name. It performs `IDN_ASCCHECK` in addition with `IDN_ENCODE_QUERY`.

`IDN_ENCODE_STORED` is used to encode a “stored string” (see the IDNA specification). It performs `IDN_ENCODE_APP` plus `IDN_UNASCHECK`.

The `idn_decodename()` function performs the reverse of `idn_encodename()`. It converts the internationalized domain name given by *from*, which is represented in a special encoding called ACE (ASCII Compatible Encoding), to the application’s local codeset and stores in *to*, whose length is specified by *tolen*. As in `idn_encodename()`, *actions* is a bitwise-OR of the following macros.

<code>IDN_DELIMMAP</code>	Delimiter mapping
<code>IDN_NAMEPREP</code>	NAMEPREP mapping, normalization, prohibited character check and bidirectional string check
<code>IDN_UNASCHECK</code>	NAMEPREP unassigned codepoint check
<code>IDN_IDNCONV</code>	UTF-8 to IDN encoding conversion
<code>IDN_RTCHECK</code>	Round trip check
<code>IDN_ASCCHECK</code>	ASCII range character check
<code>IDN_LOCALCONV</code>	Local encoding to UTF-8 conversion

Details of this decoding process can be found in the section Name Decoding.

For convenience, `IDN_DECODE_QUERY`, `IDN_DECODE_APP`, and `IDN_DECODE_STORED` macros are also provided. `IDN_DECODE_QUERY` is used to decode a “query string” (see the IDNA specification). It is equal to

```
(IDN_DELIMMAP | IDN_NAMEPREP | IDN_IDNCONV | IDN_RTCHECK | IDN_LOCALCONV)
```

`IDN_DECODE_APP` is used for ordinary application to decode a domain name. It performs `IDN_ASCCHECK` in addition to `IDN_DECODE_QUERY`. `IDN_DECODE_STORED` is used to decode a “stored string” (see the IDNA specification). It performs `IDN_DECODE_APP` plus `IDN_UNASCHECK`.

The `idn_decodename2()` function provides the same functionality as `idn_decodename()` except that character encoding of *from* is supposed to be auxencoding. If IDN encoding is Punycode and auxencoding is ISO8859-2, for example, it is assumed that the Punycode string stored in *from* is written in ISO8859-2.

idn_decodename(3EXT)

In the IDN decode procedure, IDN_NAMEPREP is done before IDN_IDNCONV, and some non-ASCII characters are converted to ASCII characters as the result of IDN_NAMEPREP. Therefore, ACE string specified by *from* might contains those non-ASCII characters. That is the reason `docode_name2()` exists.

All of these functions return an error value of type `idn_result_t`. All values other than `idn_success` indicates some kind of failure.

Name Encoding

Name encoding is a process that transforms the specified internationalized domain name to a certain string suitable for name resolution. For each label in a given domain name, the encoding processor performs:

1. Convert to UTF-8 (IDN_LOCALCONV)
Convert the encoding of the given domain name from application's local encoding (for example, ISO8859-1) to UTF-8.
2. Delimiter mapping (IDN_DELIMMAP)
Map domain name delimiters to '.' (U+002E). The recognized delimiters are: U+3002 (ideographic full stop), U+FF0E (fullwidth full stop), U+FF61 (halfwidth ideographic full stop).
3. Local mapping (IDN_LOCALMAP)
Apply character mapping whose rule is determined by the top-level domain name.
4. NAMEPREP (IDN_NAMEPREP, IDN_UNASCHECK)
Perform name preparation (NAMEPREP), which is a standard process for name canonicalization of internationalized domain names.
NAMEPREP consists of 5 steps: mapping, normalization, prohibited character check, bidirectional text check, and unassigned codepoint check. The first four steps are done by IDN_NAMEPREP, and the last step is done by IDN_UNASCHECK.
5. ASCII range character check (IDN_ASCCHECK)
Checks if the domain name contains non-LDH ASCII characters (not letter, digit, or hyphen characters), or it begins or end with hyphen.
6. Convert to ACE (IDN_IDNCONV)
Convert the NAMEPREPped name to a special encoding designed for representing internationalized domain names.
The encoding is known as ACE (ASCII Compatible Encoding) since a string in the encoding is just like a traditional ASCII domain name consisting of only letters, digits and hyphens.
7. Label length check (IDN_LENCHECK)
For each label, check the number of characters in it. It must be in the range of 1 to 63.

idn_decodename(3EXT)

Name Decoding

Name decoding is a reverse process of the name encoding. It transforms the specified internationalized domain name in a special encoding suitable for name resolution to the normal name string in the application's current codeset. However, name encoding and name decoding are not symmetric.

For each label in a given domain name, the decoding processor performs:

1. Delimiter mapping (IDN_DELIMMAP)
Map domain name delimiters to '.' (U+002E). The recognized delimiters are: U+3002 (ideographic full stop), U+FF0E (fullwidth full stop), U+FF61 (halfwidth ideographic full stop).
2. NAMEPREP (IDN_NAMEPREP, IDN_UNASCHECK)
Perform name preparation (NAMEPREP), which is a standard process for name canonicalizaion of internationalized domain names.
3. Convert to UTF-8 (IDN_IDNCONV)
Convert the encoding of the given domain name from ACE to UTF-8.
4. Round trip check (IDN_RTCHECK)
Encode the result of (3) using the Name Encoding scheme, and then compare it with the result of the step (2). If they are different, the check is failed. If IDN_UNASCHECK, IDN_ASCCHECK or both are specified, they are also done in the encoding processes.
5. Convert to local encoding
Convert the result of (3) from UTF-8 to the application's local encoding (for example, ISO8859-1).

If prohibited character check, unassigned codepoint check or bidirectional text check at step (2) failed, or if round trip check at step (4) failed, the original input label is returned.

Disabling IDN

If your application should always disable internationalized domain name support for some reason, call

```
(void) idn_enable(0);
```

before performing encoding/decoding. Afterward, you can enable the support by calling

```
(void) idn_enable(1);
```

RETURN VALUES

These functions return values of type `idn_result_t` to indicate the status of the call. The following is a complete list of the status codes.

<code>idn_success</code>	Not an error. The call succeeded.
<code>idn_notfound</code>	Specified information does not exist.
<code>idn_invalid_encoding</code>	The encoding of the specified string is invalid.

idn_decodename(3EXT)

idn_invalid_syntax	There is a syntax error in internal configuration file(s).
idn_invalid_name	The specified name is not valid.
idn_invalid_message	The specified message is not valid.
idn_invalid_action	The specified action contains invalid flags.
idn_invalid_codepoint	The specified Unicode code point value is not valid.
idn_invalid_length	The number of characters in an ACE label is not in the range of 1 to 63.
idn_buffer_overflow	The specified buffer is too small to hold the result.
idn_noentry	The specified key does not exist in the hash table.
idn_nomemory	Memory allocation using malloc failed.
idn_nofile	The specified file could not be opened.
idn_nomapping	Some characters do not have the mapping to the target character set.
idn_context_required	Context information is required.
idn_prohibited	The specified string contains some prohibited characters.
idn_failure	Generic error which is not covered by the above codes.

EXAMPLES

EXAMPLE 1 Get the address of an internationalized domain name.

To get the address of an internationalized domain name in the application's local codeset, use `idn_encodename()` to convert the name to the format suitable for passing to resolver functions.

```
#include <idn/api.h>
#include <sys/socket.h>
#include <netdb.h>

...

idn_result_t r;
char ace_name[256];
struct hostent *hp;
int error_num;

...

r = idn_encodename(IDN_ENCODE_APP, name, ace_name,
                  sizeof(ace_name));
```

idn_decodename(3EXT)

EXAMPLE 1 Get the address of an internationalized domain name. *(Continued)*

```
if (r != idn_success) {
    fprintf(stderr, gettext("idn_encodename failed.\n"));
    exit(1);
}

hp = getipnodebyname(ace_name, AF_INET6, AI_DEFAULT, &error_num);

...
```

EXAMPLE 2 Decode the internationalized domain name.

To decode the internationalized domain name returned from a resolver function, use `idn_decodename()`.

```
#include <idn/api.h>
#include <sys/socket.h>
#include <netdb.h>

...

idn_result_t r;
char local_name[256];
struct hostent *hp;
int error_num;

...

hp = getipnodebyname(name, AF_INET, AI_DEFAULT, &error_num);
if (hp != (struct hostent *)NULL) {
    r = idn_decodename(IDN_DECODE_APP, hp->h_name, local_name,
                      sizeof(local_name));
    if (r != idn_success) {
        fprintf(stderr, gettext("idn_decodename failed.\n"));
        exit(1);
    }
    printf(gettext("name: %s\n"), local_name);
}

...
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWidn1, SUNWidnd
CSI	Enabled
Interface Stability	Evolving
MT-Level	Unsafe

idn_decodename(3EXT)

SEE ALSO intro(3), libidnkit(3LIB), setlocale(3C), hosts(4), attributes(5), environ(5)

RFC 3490 Internationalizing Domain Names in Applications (IDNA)

RFC 3491 Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)

RFC 3492 Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)

RFC 3454 Preparation of Internationalized Strings ("stringprep")

RFC 952 DoD Internet Host Table Specification

RFC 921 Domain Name System Implementation Schedule - Revised

STD 3, RFC 1122 Requirements for Internet Hosts -- Communication Layers

STD 3, RFC 1123 Requirements for Internet Hosts -- Applications and Support

Unicode Standard Annex #15: Unicode Normalization Forms, Version 3.2.0.
<http://www.unicode.org/unicode/reports/tr15/tr15-22.html>

International Language Environments Guide (for this version of Solaris)

**COPYRIGHT
AND LICENSE**

Copyright (c) 2000-2002 Japan Network Information Center. All rights reserved.

By using this file, you agree to the terms and conditions set forth below.

LICENSE TERMS AND CONDITIONS

The following License Terms and Conditions apply, unless a different license is obtained from Japan Network Information Center ("JPNIC"), a Japanese association, Kokusai-Kougyou-Kanda Bldg 6F, 2-3-4 Uchi-Kanda, Chiyoda-ku, Tokyo 101-0047, Japan.

1. Use, Modification and Redistribution (including distribution of any modified or derived work) in source and/or binary forms is permitted under this License Terms and Conditions.
2. Redistribution of source code must retain the copyright notices as they appear in each source code file, this License Terms and Conditions.
3. Redistribution in binary form must reproduce the Copyright Notice, this License Terms and Conditions, in the documentation and/or other materials provided with the distribution. For the purposes of binary distribution the "Copyright Notice" refers to the following language: "Copyright (c) 2000-2002 Japan Network Information Center. All rights reserved."
4. The name of JPNIC may not be used to endorse or promote products derived from this Software without specific prior written approval of JPNIC.

idn_decodename(3EXT)

5. Disclaimer/Limitation of Liability: THIS SOFTWARE IS PROVIDED BY JPNIC "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JPNIC BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

NOTES

The `idn_nameinit()` function checks internal system configuration files such as `/etc/idn/idn.conf` and `/etc/idn/idnalias.conf` if they are in the proper access mode and ownership. If they are not in the proper access mode or ownership, the function will not read and use the configurations defined in the files but use default values. In this case the function will also issue a warning message such as:

```
idn_nameinit: warning: config file (/etc/idn/idn.conf) not in proper
                    access mode or ownership - the file ignored.
```

The proper access mode and the ownership are described in the package prototype file of `SUNWidnl`. It is also recommended not to change the system configuration files.

IFDHCloseChannel(3SMARTCARD)

NAME	IFDHCloseChannel – close the communication channel with an IFD				
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHCloseChannel (DWORD <i>Lun</i>) ;</pre>				
PARAMETERS	The IFDHCloseChannel () function takes the following parameters:				
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> </table>	<i>Lun</i>	Logical Unit Number		
<i>Lun</i>	Logical Unit Number				
DESCRIPTION	The IFDHCloseChannel () function closes the communication channel for the Interface Device (IFD) specified by <i>Lun</i> . If a smart card is present in the slot, it must be powered down. All internal resources (such as file descriptors) associated with this IFD can be freed with this function.				
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error has occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_COMMUNICATION_ERROR	An error has occurred.
IFD_SUCCESS	Successful completion.				
IFD_COMMUNICATION_ERROR	An error has occurred.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)				
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location <code>/var/sadm/pkg/SUNWocfh/install/copyright</code>. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.				

IFDHControl(3SMARTCARD)

NAME	IFDHControl – send control information to an IFD								
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHControl (DWORD <i>Lun</i>, PCHAR <i>TxBuffer</i>, DWORD <i>TxLength</i>, PCHAR <i>RxBuffer</i>, PDWORD <i>RxLength</i>);</pre>								
PARAMETERS	The IFDHControl () function takes the following parameters:								
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> <tr> <td><i>TxBuffer</i></td><td>Control bytes to send</td></tr> <tr> <td><i>TxLength</i></td><td>Length of bytes to send</td></tr> <tr> <td><i>RxLength</i></td><td>Expected length of response</td></tr> </table>	<i>Lun</i>	Logical Unit Number	<i>TxBuffer</i>	Control bytes to send	<i>TxLength</i>	Length of bytes to send	<i>RxLength</i>	Expected length of response
<i>Lun</i>	Logical Unit Number								
<i>TxBuffer</i>	Control bytes to send								
<i>TxLength</i>	Length of bytes to send								
<i>RxLength</i>	Expected length of response								
Output	<table> <tr> <td><i>RxBuffer</i></td><td>Buffer to receive response</td></tr> <tr> <td><i>RxLength</i></td><td>Length of response received</td></tr> </table>	<i>RxBuffer</i>	Buffer to receive response	<i>RxLength</i>	Length of response received				
<i>RxBuffer</i>	Buffer to receive response								
<i>RxLength</i>	Length of response received								
DESCRIPTION	The IFDHControl () performs control information exchange with some types of readers such as PIN pads, biometrics, and LCD panels according to the MCT and CTBCS specification. This function does not exchange data with the card.								
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_RESPONSE_TIMEOUT</td><td>The response has timed out.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error has occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_RESPONSE_TIMEOUT	The response has timed out.	IFD_COMMUNICATION_ERROR	An error has occurred.		
IFD_SUCCESS	Successful completion.								
IFD_RESPONSE_TIMEOUT	The response has timed out.								
IFD_COMMUNICATION_ERROR	An error has occurred.								
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Evolving								
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)								
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location /var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.								

IFDHCreateChannel(3SMARTCARD)

NAME	IFDHCreateChannel – create a communication channel with an IFD				
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHCreateChannel (DWORD <i>Lun</i>, DWORD <i>Channel</i>) ;</pre>				
PARAMETERS	The IFDHCreateChannel() function takes the following parameters:				
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> <tr> <td><i>Channel</i></td><td>Channel ID</td></tr> </table>	<i>Lun</i>	Logical Unit Number	<i>Channel</i>	Channel ID
<i>Lun</i>	Logical Unit Number				
<i>Channel</i>	Channel ID				
DESCRIPTION	The IFDHCreateChannel() function is similar to IFDHCreateChannelByName(3SMARTCARD). It takes <i>Channel</i> (a number) as an argument instead of the device name string. The <i>Channel</i> argument is typically passed from configuration information by the smart card framework server/daemon (the caller) to the IFD handler. The IFD handler can use this <i>Channel</i> appropriately to create a communication channel to the card terminal.				
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error has occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_COMMUNICATION_ERROR	An error has occurred.
IFD_SUCCESS	Successful completion.				
IFD_COMMUNICATION_ERROR	An error has occurred.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)				
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location <code>/var/sadm/pkg/SUNWocfh/install/copyright</code>. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.				

IFDHCreateChannelByName(3SMARTCARD)

NAME	IFDHCreateChannelByName – create a communication channel with an IFD				
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHCreateChannelByName(DWORD <i>Lun</i>, LPSTR <i>devicename</i>);</pre>				
PARAMETERS	The IFDHCreateChannelByName() function takes the following parameters:				
Input	<i>Lun</i> Logical Unit Number <i>devicename</i> Device name path				
DESCRIPTION	The IFDHCreateChannelByName() function opens a communication channel with a card terminal specified by <i>devicename</i>. This function can use open(2) or other system call to open the device and establish a communication path. The caller of this function (smart card framework) assigns a logical unit number <i>Lun</i> per card terminal and slot and passes this value to IFDHCreateChannelByName(). If the IFD handler supports multiple terminals with one instance of the handler (as indicated by capability TAG_IFD_SIMULTANEOUS_ACCESS), it communicates with the card terminal corresponding to this <i>Lun</i>. If the IFD handler supports only one terminal with one slot per instance, it can choose to ignore the <i>Lun</i>. The Logical Unit Number, <i>Lun</i> is encoded as 0xXXXXYYYY, where YYYY represents the lower 16 bits that correspond to the slot number for terminals with multiple slots. Most of the readers have only one slot, in which case YYYY is 0. XXXX represents the next 16 bits that correspond to the card terminal and can range between 0 and a number returned by TAG_IFD_SIMULTANEOUS_ACCESS.				
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_COMMUNICATION_ERROR	An error occurred.
IFD_SUCCESS	Successful completion.				
IFD_COMMUNICATION_ERROR	An error occurred.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				
SEE ALSO	attributes(5)				
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location				

IFDHCreateChannelByName(3SMARTCARD)

/var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.

IFDHGetCapabilities(3SMARTCARD)

NAME	IFDHGetCapabilities – get IFD capabilities						
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHGetCapabilities (DWORD <i>Lun</i>, DWORD <i>Tag</i>, PDWORD <i>Length</i>, PCHAR <i>Value</i>);</pre>						
PARAMETERS	The IFDHGetCapabilities() function takes the following parameters:						
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> <tr> <td><i>Tag</i></td><td>Tag of the desired data value</td></tr> <tr> <td><i>Length</i></td><td>Maximum length of the desired data value</td></tr> </table>	<i>Lun</i>	Logical Unit Number	<i>Tag</i>	Tag of the desired data value	<i>Length</i>	Maximum length of the desired data value
<i>Lun</i>	Logical Unit Number						
<i>Tag</i>	Tag of the desired data value						
<i>Length</i>	Maximum length of the desired data value						
Output	<table> <tr> <td><i>Length</i></td><td>Length of the data returned</td></tr> <tr> <td><i>Value</i></td><td>Value of the desired data</td></tr> </table>	<i>Length</i>	Length of the data returned	<i>Value</i>	Value of the desired data		
<i>Length</i>	Length of the data returned						
<i>Value</i>	Value of the desired data						
DESCRIPTION	The IFDHGetCapabilities() function retrieves the terminal or card capabilities for the terminal or card specified by <i>Lun</i>. The <i>Tag</i> parameter can have one of the following values: <table> <tr> <td>TAG_IFD_ATR</td><td>Return the ATR (Answer To Reset). This is the default value.</td></tr> <tr> <td>TAG_IFD_SIMULTANEOUS_ACCESS</td><td>Return the number of sessions the driver can handle. This value is used for multiple terminals sharing the same IFD handler.</td></tr> <tr> <td>TAG_IFD_SLOTS_NUMBER</td><td>Return the number of slots in this terminal.</td></tr> </table> If the TAG_IFD_SIMULTANEOUS_ACCESS and TAG_IFD_SLOTS_NUMBER tags are not supported, the error value IFD_ERROR_TAG must be returned.	TAG_IFD_ATR	Return the ATR (Answer To Reset). This is the default value.	TAG_IFD_SIMULTANEOUS_ACCESS	Return the number of sessions the driver can handle. This value is used for multiple terminals sharing the same IFD handler.	TAG_IFD_SLOTS_NUMBER	Return the number of slots in this terminal.
TAG_IFD_ATR	Return the ATR (Answer To Reset). This is the default value.						
TAG_IFD_SIMULTANEOUS_ACCESS	Return the number of sessions the driver can handle. This value is used for multiple terminals sharing the same IFD handler.						
TAG_IFD_SLOTS_NUMBER	Return the number of slots in this terminal.						
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_ERROR_TAG</td><td>An error occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_ERROR_TAG	An error occurred.		
IFD_SUCCESS	Successful completion.						
IFD_ERROR_TAG	An error occurred.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)						
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location						

IFDHGetCapabilities(3SMARTCARD)

/var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.

IFDHICCPresence(3SMARTCARD)

NAME	IFDHICCPresence – check for the presence of a smart card				
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHICCPresence (DWORD Lun) ;</pre>				
PARAMETERS	The IFDHICCPresence () function takes the following parameter:				
Input	<i>Lun</i> Logical Unit Number				
DESCRIPTION	The IFDHICCPresence () function checks for the presence of an ICC (smart card) in the reader or slot specified by <i>Lun</i> .				
RETURN VALUES	The following values are returned: IFD_ICC_PRESENT ICC is present. IFD_ICC_NOT_PRESENT ICC is not present. IFD_COMMUNICATION_ERROR An error occurred.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)				
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location /var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.				

IFDHPowerICC(3SMARTCARD)

NAME	IFDHPowerICC – power up or power down the smart card									
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHPowerICC(DWORD <i>Lun</i>, DWORD <i>Action</i>, PCHAR <i>Atr</i>, PDWORD <i>AtrLength</i>) ;</pre>									
PARAMETERS	The IFDHPowerICC() takes the following parameters:									
Input	<i>Lun</i>	Logical Unit Number								
	<i>Action</i>	Action to be taken								
	<i>AtrLength</i>	Maximum length of the ATR								
Output	<i>Atr</i>	Answer to Reset (ATR) value of the inserted card								
	<i>AtrLength</i>	Actual length of the ATR								
DESCRIPTION	The IFDHPowerICC() function controls the power and reset signals of the ICC (smart card) at the reader or slot specified by <i>Lun</i>. The <i>Action</i> parameter can take one of the following values: <table><tr><td>IFD_POWER_UP</td><td>Power and reset the card. Return the ATR and its length.</td></tr><tr><td>IFD_POWER_DOWN</td><td>Power down the card. The <i>Atr</i> and <i>AtrLength</i> parameters are set to 0.</td></tr><tr><td>IFD_RESET</td><td>Perform a quick reset on the card. Return the ATR and its length.</td></tr></table> The IFD handler caches the ATR during a power up or reset and returns the ATR and its length in a call to IFDHGetCapabilities(3SMARTCARD). Memory cards without an ATR return IFD_SUCCESS on power up or reset but <i>Atr</i> and <i>AtrLength</i> are set to 0.		IFD_POWER_UP	Power and reset the card. Return the ATR and its length.	IFD_POWER_DOWN	Power down the card. The <i>Atr</i> and <i>AtrLength</i> parameters are set to 0.	IFD_RESET	Perform a quick reset on the card. Return the ATR and its length.		
IFD_POWER_UP	Power and reset the card. Return the ATR and its length.									
IFD_POWER_DOWN	Power down the card. The <i>Atr</i> and <i>AtrLength</i> parameters are set to 0.									
IFD_RESET	Perform a quick reset on the card. Return the ATR and its length.									
RETURN VALUES	The following values are returned: <table><tr><td>IFD_SUCCESS</td><td>Successful completion.</td></tr><tr><td>IFD_ERROR_POWER_ACTION</td><td>An error occurred while powering up or resetting the card.</td></tr><tr><td>IFD_NOT_SUPPORTED</td><td>The action specified by <i>Action</i> is not supported.</td></tr><tr><td>IFD_COMMUNICATION_ERROR</td><td>An error occurred.</td></tr></table>		IFD_SUCCESS	Successful completion.	IFD_ERROR_POWER_ACTION	An error occurred while powering up or resetting the card.	IFD_NOT_SUPPORTED	The action specified by <i>Action</i> is not supported.	IFD_COMMUNICATION_ERROR	An error occurred.
IFD_SUCCESS	Successful completion.									
IFD_ERROR_POWER_ACTION	An error occurred while powering up or resetting the card.									
IFD_NOT_SUPPORTED	The action specified by <i>Action</i> is not supported.									
IFD_COMMUNICATION_ERROR	An error occurred.									
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:									

IFDHPowerICC(3SMARTCARD)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO IFDHCreateChannelByName(3SMARTCARD),
IFDHGetCapabilities(3SMARTCARD), attributes(5)

NOTES This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location
/var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.

IFDHSetCapabilities(3SMARTCARD)

NAME	IFDHSetCapabilities – set slot or card capabilities								
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHSetCapabilities (DWORD <i>Lun</i>, DWORD <i>Tag</i>, DWORD <i>Length</i>, PCHAR <i>Value</i>) ;</pre>								
PARAMETERS	The IFDHSetCapabilities() function takes the following parameters:								
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> <tr> <td><i>Tag</i></td><td>Tag of the desired data value</td></tr> <tr> <td><i>Length</i></td><td>Maximum length of the desired data value</td></tr> <tr> <td><i>Value</i></td><td>Value of the desired data</td></tr> </table>	<i>Lun</i>	Logical Unit Number	<i>Tag</i>	Tag of the desired data value	<i>Length</i>	Maximum length of the desired data value	<i>Value</i>	Value of the desired data
<i>Lun</i>	Logical Unit Number								
<i>Tag</i>	Tag of the desired data value								
<i>Length</i>	Maximum length of the desired data value								
<i>Value</i>	Value of the desired data								
DESCRIPTION	The IFDHSetCapabilities() function sets the slot or card capabilities for the slot or card specified by <i>Lun</i> .								
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_ERROR_TAG</td><td>The tag is invalid.</td></tr> <tr> <td>IFD_ERROR_SET_FAILURE</td><td>The value of the data could not be set.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error occurred.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_ERROR_TAG	The tag is invalid.	IFD_ERROR_SET_FAILURE	The value of the data could not be set.	IFD_COMMUNICATION_ERROR	An error occurred.
IFD_SUCCESS	Successful completion.								
IFD_ERROR_TAG	The tag is invalid.								
IFD_ERROR_SET_FAILURE	The value of the data could not be set.								
IFD_COMMUNICATION_ERROR	An error occurred.								
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Evolving								
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), IFDHGetCapabilities(3SMARTCARD), attributes(5)								
NOTES	This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location /var/sadm/pkg/SUNWocfh/install/copyright. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.								

IFDHSetProtocolParameters(3SMARTCARD)

NAME	IFDHSetProtocolParameters – set protocol parameters												
SYNOPSIS	<pre>#include <smartcard/ifdhandler.h> RESPONSECODE IFDHSetProtocolParameters (DWORD <i>Lun</i>, DWORD <i>Protocol</i>, UCHAR <i>Flags</i>, UCHAR <i>PTS1</i>, UCHAR <i>PTS2</i>, UCHAR <i>PTS3</i>);</pre>												
PARAMETERS	The IFDHSetProtocolParameters() function takes the following parameters:												
Input	<table> <tr> <td><i>Lun</i></td><td>Logical Unit Number</td></tr> <tr> <td><i>Protocol</i></td><td>Desired protocol</td></tr> <tr> <td><i>Flags</i></td><td>The bitwise-inclusive OR of the flags</td></tr> <tr> <td><i>PTS1</i></td><td>1st PTS Value</td></tr> <tr> <td><i>PTS2</i></td><td>2nd PTS Value</td></tr> <tr> <td><i>PTS3</i></td><td>3rd PTS Value</td></tr> </table>	<i>Lun</i>	Logical Unit Number	<i>Protocol</i>	Desired protocol	<i>Flags</i>	The bitwise-inclusive OR of the flags	<i>PTS1</i>	1st PTS Value	<i>PTS2</i>	2nd PTS Value	<i>PTS3</i>	3rd PTS Value
<i>Lun</i>	Logical Unit Number												
<i>Protocol</i>	Desired protocol												
<i>Flags</i>	The bitwise-inclusive OR of the flags												
<i>PTS1</i>	1st PTS Value												
<i>PTS2</i>	2nd PTS Value												
<i>PTS3</i>	3rd PTS Value												
DESCRIPTION	The IFDHSetProtocolParameters() function sets the Protocol Type Selection (PTS) of the slot or card using the three PTS values as defined in ISO 7816. The <i>Protocol</i> parameter can take an integer value between 0 and 14, inclusive, corresponding to the protocol T=0, T=1, ..., T=14. The <i>Flags</i> parameter can have the value of one of the following or the bitwise-inclusive OR of two or more of the following: <table> <tr> <td>IFD_NEGOTIATE_PTS1</td><td>Use the <i>PTS1</i> value.</td></tr> <tr> <td>IFD_NEGOTIATE_PTS2</td><td>Use the <i>PTS2</i> value.</td></tr> <tr> <td>IFD_NEGOTIATE_PTS3</td><td>Use the <i>PTS3</i> value.</td></tr> </table>	IFD_NEGOTIATE_PTS1	Use the <i>PTS1</i> value.	IFD_NEGOTIATE_PTS2	Use the <i>PTS2</i> value.	IFD_NEGOTIATE_PTS3	Use the <i>PTS3</i> value.						
IFD_NEGOTIATE_PTS1	Use the <i>PTS1</i> value.												
IFD_NEGOTIATE_PTS2	Use the <i>PTS2</i> value.												
IFD_NEGOTIATE_PTS3	Use the <i>PTS3</i> value.												
RETURN VALUES	The following values are returned: <table> <tr> <td>IFD_SUCCESS</td><td>Successful completion.</td></tr> <tr> <td>IFD_COMMUNICATION_ERROR</td><td>An error occurred.</td></tr> <tr> <td>IFD_ERROR_PTS_FAILURE</td><td>The PTS value could not be set.</td></tr> <tr> <td>IFD_PROTOCOL_NOT_SUPPORTED</td><td>The protocol is not supported.</td></tr> </table>	IFD_SUCCESS	Successful completion.	IFD_COMMUNICATION_ERROR	An error occurred.	IFD_ERROR_PTS_FAILURE	The PTS value could not be set.	IFD_PROTOCOL_NOT_SUPPORTED	The protocol is not supported.				
IFD_SUCCESS	Successful completion.												
IFD_COMMUNICATION_ERROR	An error occurred.												
IFD_ERROR_PTS_FAILURE	The PTS value could not be set.												
IFD_PROTOCOL_NOT_SUPPORTED	The protocol is not supported.												
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving								
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)												

NOTES | This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface is available at the default location `/var/sadm/pkg/SUNWocfh/install/copyright`. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.

IFDHTransmitToICC(3SMARTCARD)

NAME	IFDHTransmitToICC – transmit APDU to a smart card											
SYNOPSIS	#include <smartcard/ifdhandler.h> RESPONSECODE IFDHTransmitToICC (DWORD <i>Lun</i> , SCARD_IO_HEADER <i>SendPci</i> , PUCHAR <i>TxBuffer</i> , DWORD <i>TxLength</i> , PCHAR <i>RxBuffer</i> , PDWORD <i>RxLength</i> , PSCARD_IO_HEADER <i>RecvPci</i>) ;											
PARAMETERS	The IFDHTransmitToICC() function takes the following parameters:											
Input	<i>Lun</i>	Logical Unit Number										
	<i>SendPci</i>	Send-Protocol structure										
	<i>TxBuffer</i>	Buffer containing the APDU to be sent										
	<i>TxLength</i>	Length of sent APDU										
Output	<i>RxBuffer</i>	Received buffer for response APDU										
	<i>RxLength</i>	Length of APDU-response										
	<i>RecvPci</i>	Receive-Protocol structure										
DESCRIPTION	The IFDHTransmitToICC() function performs an Application Protocol Data Unit (APDU) exchange with the card or slot specified by <i>Lun</i>. The IFD handler is responsible for performing any protocol-specific (such as T0/T1) APDU exchanges with the card. The Protocol structure SCARD_IO_HEADER contains the following members: <table><tr><td>Protocol</td><td>Values range from 0 through 14, inclusive, indicating protocol T=0, T=1, ..., T=14.</td></tr><tr><td>Length</td><td>Not used.</td></tr></table>		Protocol	Values range from 0 through 14, inclusive, indicating protocol T=0, T=1, ..., T=14.	Length	Not used.						
Protocol	Values range from 0 through 14, inclusive, indicating protocol T=0, T=1, ..., T=14.											
Length	Not used.											
RETURN VALUES	The following values are returned: <table><tr><td>IFD_SUCCESS</td><td>Successful completion.</td></tr><tr><td>IFD_RESPONSE_TIMEOUT</td><td>The response timed out.</td></tr><tr><td>IFD_ICC_NOT_PRESENT</td><td>The card is not present.</td></tr><tr><td>IFD_PROTOCOL_NOT_SUPPORTED</td><td>The protocol is not supported</td></tr><tr><td>IFD_COMMUNICATION_ERROR</td><td>An error occurred.</td></tr></table>		IFD_SUCCESS	Successful completion.	IFD_RESPONSE_TIMEOUT	The response timed out.	IFD_ICC_NOT_PRESENT	The card is not present.	IFD_PROTOCOL_NOT_SUPPORTED	The protocol is not supported	IFD_COMMUNICATION_ERROR	An error occurred.
IFD_SUCCESS	Successful completion.											
IFD_RESPONSE_TIMEOUT	The response timed out.											
IFD_ICC_NOT_PRESENT	The card is not present.											
IFD_PROTOCOL_NOT_SUPPORTED	The protocol is not supported											
IFD_COMMUNICATION_ERROR	An error occurred.											
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:											
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving						
ATTRIBUTE TYPE	ATTRIBUTE VALUE											
Interface Stability	Evolving											
SEE ALSO	IFDHCreateChannelByName(3SMARTCARD), attributes(5)											

NOTES | This manual page is derived from the MUSCLE PC/SC IFD Driver Developer Kit documentation. License terms and attribution and copyright information for this interface are available at the default location `/var/sadm/pkg/SUNWocfh/install/copyright`. If the Solaris Operating Environment has been installed anywhere other than the default location, modify the path to access the file at the installed location.

ilogb(3M)

NAME	ilogb – returns an unbiased exponent				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> int ilogb(double x);</pre>				
DESCRIPTION	The <code>ilogb()</code> function returns the exponent part of x . Formally, the return value is the integral part of $\log_r x $ as a signed integral value, for non-zero finite x , where r is the radix of the machine's floating point arithmetic.				
RETURN VALUES	Upon successful completion, <code>ilogb()</code> returns the exponent part of x. If x is 0, <code>ilogb()</code> returns <code>-INT_MAX</code>. If x is NaN or $\pm\text{Inf}$, <code>ilogb()</code> returns <code>INT_MAX</code>.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>logb(3M)</code> , <code>attributes(5)</code>				

NAME	isencrypt – determine whether a buffer of characters is encrypted				
SYNOPSIS	<pre>cc [flag...] [file...] -lgen [library...] #include<libgen.h> int isencrypt(const char *fbuf, size_t ninbuf);</pre>				
DESCRIPTION	isencrypt () uses heuristics to determine whether a buffer of characters is encrypted. It requires two arguments: a pointer to an array of characters and the number of characters in the buffer. isencrypt () assumes that the file is not encrypted if all the characters in the first block are ASCII characters. If there are non-ASCII characters in the first <i>ninbuf</i> characters, and if the setlocale () LC_CTYPE category is set to C or ascii, isencrypt () assumes that the buffer is encrypted If the LC_CTYPE category is set to a value other than C or ascii, then isencrypt () uses a combination of heuristics to determine if the buffer is encrypted. If <i>ninbuf</i> has at least 64 characters, a chi-square test is used to determine if the bytes in the buffer have a uniform distribution; if it does, then isencrypt () assumes the buffer is encrypted. If the buffer has less than 64 characters, a check is made for null characters and a terminating new-line to determine whether the buffer is encrypted.				
RETURN VALUES	If the buffer is encrypted, 1 is returned; otherwise, zero is returned.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	setlocale(3C), attributes(5)				
NOTES	When compiling multithreaded applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multithreaded applications.				

isnan(3M)

NAME `isnan` – test for NaN

SYNOPSIS `cc [ flag ... ] file ... -lm [ library ... ]`
 `#include <math.h>`

 `int isnan(double x);`

DESCRIPTION The `isnan()` function tests whether *x* is NaN.

RETURN VALUES The `isnan()` function returns non-zero if *x* is NaN. Otherwise, 0 is returned.

USAGE On systems not supporting NaN, `isnan()` always returns 0.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `attributes(5)`

NAME	j0, j1, jn – Bessel functions of the first kind				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double j0(double x); double j1(double x); double jn(int n, double x);</pre>				
DESCRIPTION	The j0(), j1() and jn() functions compute Bessel functions of x of the first kind of orders 0, 1 and n respectively.				
RETURN VALUES	Upon successful completion, j0(), j1() and jn() return the relevant Bessel value of x of the first kind. If the x argument is too large in magnitude, 0 is returned and <code>errno</code> may be set to <code>ERANGE</code>. If x is NaN, NaN is returned. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The j0(), j1() and jn() functions may fail if: <code>ERANGE</code> The value of x was too large in magnitude.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling j0(), j1() or jn(). If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>y0(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

kstat(3KSTAT)

NAME	kstat – kernel statistics facility				
DESCRIPTION	The <code>kstat</code> facility is a general-purpose mechanism for providing kernel statistics to users.				
The kstat model	The kernel maintains a linked list of statistics structures, or <code>kstats</code> . Each <code>kstat</code> has a common header section and a type-specific data section. The header section is defined by the <code>kstat_t</code> structure:				
kstat header	<pre>typedef intkid_t; /* unique kstat id */ typedef struct kstat { /* * Fields relevant to both kernel and user */ hrtime_t ks_crttime; /* creation time */ struct kstat *ks_next; /* kstat chain linkage */ kid_t ks_kid; /* unique kstat ID */ char ks_module[KSTAT_STRLEN]; /* module name */ uchar_t ks_resv; /* reserved */ int ks_instance; /* module's instance */ char ks_name[KSTAT_STRLEN]; /* kstat name */ uchar_t ks_type; /* kstat data type */ char ks_class[KSTAT_STRLEN]; /* kstat class */ uchar_t ks_flags; /* kstat flags */ void *ks_data; /* kstat type-specific data */ uint_t ks_ndata; /* # of data records */ size_t ks_data_size; /* size of kstat data section */ hrtime_t ks_snaptime; /* time of last data snapshot */ /* * Fields relevant to kernel only */ int(*ks_update)(struct kstat *, int); void *ks_private; int(*ks_snapshot)(struct kstat *, void *, int); void *ks_lock; } kstat_t;</pre> The fields that are of significance to the user are: <table> <tr> <td><code>ks_crttime</code></td><td>The time the <code>kstat</code> was created. This allows you to compute the rates of various counters since the <code>kstat</code> was created; "rate since boot" is replaced by the more general concept of "rate since <code>kstat</code> creation". All times associated with <code>kstats</code> (such as creation time, last snapshot time, <code>kstat_timer_t</code> and <code>kstat_io_t</code> timestamps, and the like) are 64-bit nanosecond values. The accuracy of <code>kstat</code> timestamps is machine dependent, but the precision (units) is the same across all platforms. See <code>gethrtime(3C)</code> for general information about high-resolution timestamps.</td></tr> <tr> <td><code>ks_next</code></td><td><code>kstats</code> are stored as a linked list, or chain. <code>ks_next</code> points to the next <code>kstat</code> in the chain.</td></tr> </table>	<code>ks_crttime</code>	The time the <code>kstat</code> was created. This allows you to compute the rates of various counters since the <code>kstat</code> was created; "rate since boot" is replaced by the more general concept of "rate since <code>kstat</code> creation". All times associated with <code>kstats</code> (such as creation time, last snapshot time, <code>kstat_timer_t</code> and <code>kstat_io_t</code> timestamps, and the like) are 64-bit nanosecond values. The accuracy of <code>kstat</code> timestamps is machine dependent, but the precision (units) is the same across all platforms. See <code>gethrtime(3C)</code> for general information about high-resolution timestamps.	<code>ks_next</code>	<code>kstats</code> are stored as a linked list, or chain. <code>ks_next</code> points to the next <code>kstat</code> in the chain.
<code>ks_crttime</code>	The time the <code>kstat</code> was created. This allows you to compute the rates of various counters since the <code>kstat</code> was created; "rate since boot" is replaced by the more general concept of "rate since <code>kstat</code> creation". All times associated with <code>kstats</code> (such as creation time, last snapshot time, <code>kstat_timer_t</code> and <code>kstat_io_t</code> timestamps, and the like) are 64-bit nanosecond values. The accuracy of <code>kstat</code> timestamps is machine dependent, but the precision (units) is the same across all platforms. See <code>gethrtime(3C)</code> for general information about high-resolution timestamps.				
<code>ks_next</code>	<code>kstats</code> are stored as a linked list, or chain. <code>ks_next</code> points to the next <code>kstat</code> in the chain.				

	<code>ks_kid</code> A unique identifier for the kstat. <code>ks_module,</code> <code>ks_instance</code> contain the name and instance of the the module that created the kstat. In cases where there can only be one instance, <code>ks_instance</code> is 0. <code>ks_name</code> gives a meaningful name to a kstat. The full kstat namespace is <code><ks_module,ks_instance,ks_name></code>, so the name only need be unique within a module. <code>ks_type</code> The type of data in this kstat. kstat data types are discussed below. <code>ks_class</code> Each kstat can be characterized as belonging to some broad class of statistics, such as disk, tape, net, vm, and streams. This field can be used as a filter to extract related kstats. The following values are currently in use: <code>disk</code>, <code>tape</code>, <code>controller</code>, <code>net</code>, <code>rpc</code>, <code>vm</code>, <code>kvm</code>, <code>hat</code>, <code>streams</code>, <code>kmem</code>, <code>kmem_cache</code>, <code>kstat</code>, and <code>misc</code>. (The kstat class encompasses things like <i>kstat_types</i>.) <code>ks_data,</code> <code>ks_ndata,</code> <code>ks_data_size</code> <code>ks_data</code> is a pointer to the kstat's data section. The type of data stored there depends on <code>ks_type</code>. <code>ks_ndata</code> indicates the number of data records. Only some kstat types support multiple data records. Currently, <code>KSTAT_TYPE_RAW</code>, <code>KSTAT_TYPE_NAMED</code> and <code>KSTAT_TYPE_TIMER</code> kstats support multiple data records. <code>KSTAT_TYPE_INTR</code> and <code>KSTAT_TYPE_IO</code> kstats support only one data record. <code>ks_data_size</code> is the total size of the data section, in bytes. <code>ks_snaptime</code> The timestamp for the last data snapshot. This allows you to compute activity rates: $\text{rate} = (\text{new\_count} - \text{old\_count}) / (\text{new\_snaptime} - \text{old\_snaptime});$
kstat data types	The following types of kstats are currently available: <pre>#define KSTAT_TYPE_RAW 0 /* can be anything */ #define KSTAT_TYPE_NAMED 1 /* name/value pairs */ #define KSTAT_TYPE_INTR 2 /* interrupt statistics */ #define KSTAT_TYPE_IO 3 /* I/O statistics */ #define KSTAT_TYPE_TIMER 4 /* event timers */</pre> To get a list of all kstat types currently supported in the system, tools can read out the standard system kstat <i>kstat_types</i> (full name spec is <code><"unix", 0, "kstat_types"></code>). This is a <code>KSTAT_TYPE_NAMED</code> kstat in which the <i>name</i> field describes the type of kstat, and the <i>value</i> field is the kstat type number (for example, <code>KSTAT_TYPE_IO</code> is type 3 -- see above).
Raw kstat	<code>KSTAT_TYPE_RAW</code> raw data

kstat(3KSTAT)

The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like *sysinfo*.

Name=value kstat

KSTAT_TYPE_NAMED A list of arbitrary *name=value* statistics.

```
typedef struct kstat_named {
    charname[KSTAT_STRLEN];      /* name of counter */
    uchar_t data_type;           /* data type */
    union {
        charc[16];               /* enough for 128-bit ints */
        struct {
            union {
                char *ptr;        /* NULL-terminated string */
            } addr;
            uint32_t len;         /* length of string */
        } string;
        int32_t i32;
        uint32_t ui32;
        int64_t i64;
        uint64_t ui64;

        /* These structure members are obsolete */

        int32_t l;
        uint32_t ul;
        int64_t ll;
        uint64_t ull;
    } value;                     /* value of counter */
} kstat_named_t;

/* The following types are Stable */

KSTAT_DATA_CHAR
KSTAT_DATA_INT32
KSTAT_DATA_LONG
KSTAT_DATA_UINT32
KSTAT_DATA_ULONG
KSTAT_DATA_INT64
KSTAT_DATA_UINT64

/* The following type is Evolving */

KSTAT_DATA_STRING

/* The following types are Obsolete */

KSTAT_DATA_LONGLONG
KSTAT_DATA_ULONGLONG
KSTAT_DATA_FLOAT
KSTAT_DATA_DOUBLE
```

Some devices need to publish strings that exceed the maximum value for KSTAT_DATA_CHAR in length; KSTAT_DATA_STRING is a data type that allows arbitrary-length strings to be associated with a named kstat. The macros below are the supported means to read the pointer to the string and its length.

	<pre> #define KSTAT_NAMED_STR_PTR(knptr) ((knptr)->value.string.addr.ptr) #define KSTAT_NAMED_STR_BUFLLEN(knptr) ((knptr)->value.string.len) KSTAT_NAMED_STR_BUFLLEN() returns the number of bytes required to store the string pointed to by KSTAT_NAMED_STR_PTR(); that is, strlen (KSTAT_NAMED_STR_PTR()) + 1. </pre>
Interrupt kstat	KSTAT_TYPE_INTR Interrupt statistics. An interrupt is a hard interrupt (sourced from the hardware device itself), a soft interrupt (induced by the system via the use of some system interrupt source), a watchdog interrupt (induced by a periodic timer call), spurious (an interrupt entry point was entered but there was no interrupt to service), or multiple service (an interrupt was detected and serviced just prior to returning from any of the other types). <pre> #define KSTAT_INTR_HARD 0 #define KSTAT_INTR_SOFT 1 #define KSTAT_INTR_WATCHDOG 2 #define KSTAT_INTR_SPURIOUS 3 #define KSTAT_INTR_MULTSVCS 4 #define KSTAT_NUM_INTRS 5 typedef struct kstat_intr { uint_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */ } kstat_intr_t; </pre>
Event timer kstat	KSTAT_TYPE_TIMER Event timer statistics. These provide basic counting and timing information for any type of event. <pre> typedef struct kstat_timer { char name[KSTAT_STRLEN]; /* event name */ uchar_t resv; /* reserved */ u_longlong_t num_events; /* number of events */ hrtime_t elapsed_time; /* cumulative elapsed time */ hrtime_t min_time; /* shortest event duration */ hrtime_t max_time; /* longest event duration */ hrtime_t start_time; /* previous event start time */ hrtime_t stop_time; /* previous event stop time */ } kstat_timer_t; </pre>
I/O kstat	KSTAT_TYPE_IO I/O statistics. <pre> typedef struct kstat_io { /* * Basic counters. */ u_longlong_t nread; /* number of bytes read */ u_longlong_t nwritten; /* number of bytes written */ uint_t reads; /* number of read operations */ uint_t writes; /* number of write operations */ /* </pre>

kstat(3KSTAT)

```

* Accumulated time and queue length statistics.
*
* Time statistics are kept as a running sum of "active" time.
* Queue length statistics are kept as a running sum of the
* product of queue length and elapsed time at that length --
* that is, a Riemann sum for queue length integrated against time.
*
*      ^
*      |
*      8      | i4 |
*      |      | |
* Queue 6      | |
* Length |      | |
* 4 | i2 |      | |
* | | i3 |
* 2      |
* | i1 |
* |_____|
* Time-> t1 t2 t3 t4
*
* At each change of state (entry or exit from the queue),
* we add the elapsed time (since the previous state change)
* to the active time if the queue length was non-zero during
* that interval; and we add the product of the elapsed time
* times the queue length to the running length*time sum.
*
* This method is generalizable to measuring residency
* in any defined system: instead of queue lengths, think
* of "outstanding RPC calls to server X".
*
* A large number of I/O subsystems have at least two basic
* "lists" of transactions they manage: one for transactions
* that have been accepted for processing but for which processing
* has yet to begin, and one for transactions which are actively
* being processed (but not done). For this reason, two cumulative
* time statistics are defined here: pre-service (wait) time,
* and service (run) time.
*
* The units of cumulative busy time are accumulated nanoseconds.
* The units of cumulative length*time products are elapsed time
* times queue length.
*/
hrtime_t  wtime;      /* cumulative wait (pre-service) time */
hrtime_t  wlentime;    /* cumulative wait length*time product */
hrtime_t  wlastupdate; /* last time wait queue changed */
hrtime_t  rtime;      /* cumulative run (service) time */
hrtime_t  rlentime;    /* cumulative run length*time product */

```

```

hrtime_t rlastupdate; /* last time run queue changed */
uint_t   wcnt;        /* count of elements in wait state */
uint_t   rcnt;        /* count of elements in run state */
} kstat_io_t;

```

Using libkstat

The kstat library, `libkstat`, defines the user interface (API) to the system's kstat facility.

You begin by opening `libkstat` with `kstat_open(3KSTAT)`, which returns a pointer to a fully initialized kstat control structure. This is your ticket to subsequent `libkstat` operations:

```

typedef struct kstat_ctl {
    kid_t      kc_chain_id; /* current kstat chain ID */
    kstat_t    *kc_chain;   /* pointer to kstat chain */
    int        kc_kd;       /* /dev/kstat descriptor */
} kstat_ctl_t;

```

Only the first two fields, `kc_chain_id` and `kc_chain`, are of interest to `libkstat` clients. (`kc_kd` is the descriptor for `/dev/kstat`, the kernel statistics driver. `libkstat` functions are built on top of `/dev/kstat ioctl(2)` primitives. Direct interaction with `/dev/kstat` is strongly discouraged, since it is *not* a public interface.)

`kc_chain` points to your copy of the kstat chain. You typically walk the chain to find and process a certain kind of kstat. For example, to display all I/O kstats:

```

kstat_ctl_t    *kc;
kstat_t        *ksp;
kstat_io_t      kio;

kc = kstat_open();
for (ksp = kc->kc_chain; ksp != NULL; ksp = ksp->ks_next) {
    if (ksp->ks_type == KSTAT_TYPE_IO) {
        kstat_read(kc, ksp, &kio);
        my_io_display(kio);
    }
}

```

`kc_chain_id` is the kstat chain ID, or KCID, of your copy of the kstat chain. See `kstat_chain_update(3KSTAT)` for an explanation of KCIDs.

FILES

```

/dev/kstat                kernel statistics driver

/usr/include/kstat.h

/usr/include/sys/kstat.h

```

SEE ALSO

`ioctl(2)`, `gethrtime(3C)`, `getloadavg(3C)`, `kstat_chain_update(3KSTAT)`, `kstat_close(3KSTAT)`, `kstat_data_lookup(3KSTAT)`, `kstat_lookup(3KSTAT)`, `kstat_open(3KSTAT)`, `kstat_read(3KSTAT)`, `kstat_write(3KSTAT)`, `attributes(5)`

kstat_chain_update(3KSTAT)

NAME	kstat_chain_update – update the kstat header chain				
SYNOPSIS	<pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kid_t kstat_chain_update(kstat_ctl_t *kc);</pre>				
DESCRIPTION	The <code>kstat_chain_update()</code> function brings the user's kstat header chain in sync with that of the kernel. The kstat chain is a linked list of kstat headers (<code>kstat_t</code>'s) pointed to by <code>kc->kc_chain</code>, which is initialized by <code>kstat_open(3KSTAT)</code>. This chain constitutes a list of all kstats currently in the system. During normal operation, the kernel creates new kstats and delete old ones as various device instances are added and removed, thereby causing the user's copy of the kstat chain to become out of date. The <code>kstat_chain_update()</code> function detects this condition by comparing the kernel's current kstat chain ID(KCID), which is incremented every time the kstat chain changes, to the user's KCID, <code>kc->kc_chain_id</code>. If the KCIDs match, <code>kstat_chain_update()</code> does nothing. Otherwise, it deletes any invalid kstat headers from the user's kstat chain, adds any new ones, and sets <code>kc->kc_chain_id</code> to the new KCID. All other kstat headers in the user's kstat chain are unmodified.				
RETURN VALUES	The <code>kstat_chain_update()</code> function returns the new KCID if the kstat chain has changed, 0 if it hasn't, or -1 on failure.				
FILES	/dev/kstat kernel statistics driver				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>kstat(3KSTAT)</code> , <code>kstat_close(3KSTAT)</code> , <code>kstat_data_lookup(3KSTAT)</code> , <code>kstat_lookup(3KSTAT)</code> , <code>kstat_open(3KSTAT)</code> , <code>kstat_read(3KSTAT)</code> , <code>kstat_write(3KSTAT)</code> , <code>attributes(5)</code>				

NAME	kstat_lookup, kstat_data_lookup – find a kstat by name				
SYNOPSIS	<pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kstat_t *kstat_lookup(kstat_ctl_t *kc, char *ks_module, int ks_instance, char *ks_name); void *kstat_data_lookup(kstat_t *ksp, char *name);</pre>				
DESCRIPTION	The <code>kstat_lookup()</code> function traverses the kstat chain, <code>kc->kc_chain</code>, searching for a kstat with the same <code>ks_module</code>, <code>ks_instance</code>, and <code>ks_name</code> fields; this triplet uniquely identifies a kstat. If <code>ks_module</code> is NULL, <code>ks_instance</code> is -1, or <code>ks_name</code> is NULL, then those fields will be ignored in the search. For example, <code>kstat_lookup(kc, NULL, -1, "foo")</code> will simply find the first kstat with name "foo". The <code>kstat_data_lookup()</code> function searches the kstat's data section for the record with the specified <code>name</code>. This operation is valid only for kstat types which have named data records. Currently, only the <code>KSTAT_TYPE_NAMED</code> and <code>KSTAT_TYPE_TIMER</code> kstats have named data records.				
RETURN VALUES	The <code>kstat_lookup()</code> function returns a pointer to the requested kstat if it is found, or NULL if it is not. The <code>kstat_data_lookup()</code> function returns a pointer to the requested data record if it is found. If the requested record is not found, or if the kstat type is invalid, <code>kstat_data_lookup()</code> returns NULL.				
FILES	/dev/kstat kernel statistics driver				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>kstat(3KSTAT)</code>, <code>kstat_chain_update(3KSTAT)</code>, <code>kstat_close(3KSTAT)</code>, <code>kstat_open(3KSTAT)</code>, <code>kstat_read(3KSTAT)</code>, <code>kstat_write(3KSTAT)</code>, <code>attributes(5)</code>				

kstat_open(3KSTAT)

NAME	kstat_open, kstat_close – initialize kernel statistics facility				
SYNOPSIS	<pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kstat_ctl_t *kstat_open(void) ; int kstat_close(kstat_ctl_t *kc) ;</pre>				
DESCRIPTION	The <code>kstat_open()</code> function initializes a kstat control structure, which provides access to the kernel statistics library. It returns a pointer to this structure, which must be supplied as the <code>kc</code> argument in subsequent <code>libkstat</code> function calls. The <code>kstat_close()</code> function frees all resources that were associated with <code>kc</code>. This is done automatically on <code>exit(2)</code> and <code>execve()</code> (see <code>exec(2)</code>).				
RETURN VALUES	The <code>kstat_open()</code> function returns a pointer to a kstat control structure. On failure, it returns <code>NULL</code> and no resources are allocated. The <code>kstat_close()</code> function returns 0 on success and -1 on failure.				
FILES	/dev/kstat kernel statistics driver				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>kstat(3KSTAT)</code> , <code>kstat_chain_update(3KSTAT)</code> , <code>kstat_data_lookup(3KSTAT)</code> , <code>kstat_lookup(3KSTAT)</code> , <code>kstat_read(3KSTAT)</code> , <code>kstat_write(3KSTAT)</code> , <code>attributes(5)</code>				

NAME	kstat_read, kstat_write – read or write kstat data				
SYNOPSIS	<pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kid_t kstat_read(kstat_ctl_t *kc, kstat_t *ksp, void *buf); kid_t kstat_write(kstat_ctl_t *kc, kstat_t *ksp, void *buf);</pre>				
DESCRIPTION	The <code>kstat_read()</code> function gets data from the kernel for the kstat pointed to by <code>ksp</code>. <code>ksp->ks_data</code> is automatically allocated (or reallocated) to be large enough to hold all of the data. <code>ksp->ks_ndata</code> is set to the number of data fields, <code>ksp->ks_data_size</code> is set to the total size of the data, and <code>ksp->ks_snaptime</code> is set to the high-resolution time at which the data snapshot was taken. If <code>buf</code> is non-NULL, the data is copied from <code>ksp->ks_data</code> into <code>buf</code>. The <code>kstat_write()</code> function writes data from <code>buf</code>, or from <code>ksp->ks_data</code> if <code>buf</code> is NULL, to the corresponding kstat in the kernel. Only the superuser can use <code>kstat_write()</code>.				
RETURN VALUES	On success, <code>kstat_read()</code> and <code>kstat_write()</code> return the current kstat chain ID (KCID). On failure, they return -1.				
FILES	/dev/kstat kernel statistics driver				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_close(3KSTAT), kstat_data_lookup(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT), attributes(5)				

kva_match(3SECDB)

NAME	kva_match – look up a key in a key-value array				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> - lsecdb [<i>library...</i>] #include <secdb.h> char *kva_match(kva_t *kva, char *key);</pre>				
DESCRIPTION	The kva_match() function searches a kva_t structure, which is part of the authattr_t, execattr_t, profattr_t, or userattr_t structures. The function takes two arguments: a pointer to a key value array, and a key. If the key is in the array, the function returns a pointer to the first corresponding value that matches that key. Otherwise, the function returns NULL.				
RETURN VALUES	Upon successful completion, the function returns a pointer to the value sought. Otherwise, it returns NULL.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	getauthattr(3SECDB), getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB)				
NOTES	The kva_match() function returns a pointer to data that already exists in the key-value array. It does not allocate its own memory for this pointer but obtains it from the key-value array that is passed as its first argument.				

NAME	kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process
SYNOPSIS	<pre> #include <kvm.h> #include <sys/param.h> #include <sys/user.h> #include <sys/proc.h> struct user *kvm_getu(kvm_t *kd, struct proc *proc); int kvm_getcmd(kvm_t *kd, struct proc *proc, struct user *u, char ***arg, char ***env); </pre>
DESCRIPTION	
kvm_getu()	The <code>kvm_getu()</code> function reads the u-area of the process specified by <code>proc</code> to an area of static storage associated with <code>kd</code> and returns a pointer to it. Subsequent calls to <code>kvm_getu()</code> will overwrite this static area. The <code>kd</code> argument is a pointer to a kernel descriptor returned by <code>kvm_open(3KVM)</code>. The <code>proc</code> argument is a pointer to a copy in the current process' address space of a <code>proc</code> structure, obtained, for instance, by a prior <code>kvm_nextproc(3KVM)</code> call.
kvm_getcmd()	The <code>kvm_getcmd()</code> function constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by <code>proc</code>. The <code>kd</code> argument is a pointer to a kernel descriptor returned by <code>kvm_open(3KVM)</code>. The <code>u</code> argument is a pointer to a copy in the current process' address space of a <code>user</code> structure, obtained, for instance, by a prior <code>kvm_getu()</code> call. If <code>arg</code> is not <code>NULL</code>, the command line arguments are formed into a null-terminated array of string pointers. The address of the first such pointer is returned in <code>arg</code>. If <code>env</code> is not <code>NULL</code>, then the environment is formed into a null-terminated array of string pointers. The address of the first of these is returned in <code>env</code>. The pointers returned in <code>arg</code> and <code>env</code> refer to data allocated by <code>malloc(3C)</code> and should be freed by a call to <code>free()</code> when no longer needed. See <code>malloc(3C)</code>. Both the string pointers and the strings themselves are deallocated when freed. Since the environment and command line arguments may have been modified by the user process, there is no guarantee that it will be possible to reconstruct the original command at all. Thus, <code>kvm_getcmd()</code> will make the best attempt possible, returning <code>-1</code> if the user process data is unrecognizable.
RETURN VALUES	On success, <code>kvm_getu()</code> returns a pointer to a copy of the u-area of the process specified by <code>proc</code>. On failure, it returns <code>NULL</code>. The <code>kvm_getcmd()</code> function returns <code>0</code> on success and <code>-1</code> on failure.

kvm_getu(3KVM)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `kvm_nextproc(3KVM)`, `kvm_open(3KVM)`, `kvm_read(3KVM)`, `malloc(3C)`, `libkvm(3LIB)`, `attributes(5)`

NOTES If `kvm_getcmd()` returns `-1`, the caller still has the option of using the command line fragment that is stored in the u-area.

On systems that support both 32-bit and 64-bit processes, the 64-bit implementation of `libkvm` ensures that the *arg* and *env* pointer arrays for `kvm_getcmd()` are translated to the same form as if they were 64-bit processes. Applications that wish to access the raw 32-bit stack directly can use `kvm_uread()`. See `kvm_read(3KVM)`.

NAME	kvm_nextproc, kvm_getproc, kvm_setproc – read system process structures
SYNOPSIS	<pre> #include <kvm.h> #include <sys/param.h> #include <sys/time.h> #include <sys/proc.h> struct proc *kvm_nextproc(kvm_t *kd); int kvm_setproc(kvm_t *kd); struct proc *kvm_getproc(kvm_t *kd, pid_t pid); </pre>
DESCRIPTION	
kvm_nextproc()	The <code>kvm_nextproc()</code> function may be used to sequentially read all of the system process structures from the kernel identified by <i>kd</i> (see <code>kvm_open(3KVM)</code>). Each call to <code>kvm_nextproc()</code> returns a pointer to the static memory area that contains a copy of the next valid process table entry. There is no guarantee that the data will remain valid across calls to <code>kvm_nextproc()</code>, <code>kvm_setproc()</code>, or <code>kvm_getproc()</code>. Therefore, if the process structure must be saved, it should be copied to non-volatile storage. For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to <code>kvm_nextproc()</code>, and since the cache may contain obsolete information, there is no guarantee that <i>every</i> process structure returned refers to an active process, nor is it certain that <i>all</i> processes will be reported.
kvm_setproc()	The <code>kvm_setproc()</code> function rewinds the process list, enabling <code>kvm_nextproc()</code> to rescan from the beginning of the system process table. This function will always flush the process structure cache, allowing an application to re-scan the process table of a running system.
kvm_getproc()	The <code>kvm_getproc()</code> function locates the <code>proc</code> structure of the process specified by <i>pid</i> and returns a pointer to it. This function does not interact with the process table pointer manipulated by <code>kvm_nextproc()</code>; however, the restrictions regarding the validity of the data still apply.
RETURN VALUES	On success, <code>kvm_nextproc()</code> returns a pointer to a copy of the next valid process table entry. On failure, it returns <code>NULL</code>. On success, <code>kvm_getproc()</code> returns a pointer to the <code>proc</code> structure of the process specified by <i>pid</i>. On failure, it returns <code>NULL</code>. The <code>kvm_setproc()</code> function returns 0 on success –1 on failure.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

kvm_nextproc(3KVM)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `kvm_getu(3KVM)`, `kvm_open(3KVM)`, `kvm_read(3KVM)`, `attributes(5)`

NAME	kvm_nlist – get entries from kernel symbol table				
SYNOPSIS	<pre>#include <kvm.h> #include <nlist.h> int kvm_nlist(kvm_t *kd, struct nlist *nl);</pre>				
DESCRIPTION	<code>kvm_nlist()</code> examines the symbol table from the kernel image identified by <i>kd</i> (see <code>kvm_open(3KVM)</code>) and selectively extracts a list of values and puts them in the array of <code>nlist</code> structures pointed to by <code>nl</code>. The name list pointed to by <code>nl</code> consists of an array of structures containing names, types and values. The <i>n_name</i> field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a null string) in the <i>n_name</i> field. For each entry in <code>nl</code>, if the named symbol is present in the kernel symbol table, its value and type are placed in the <i>n_value</i> and <i>n_type</i> fields. If a symbol cannot be located, the corresponding <i>n_type</i> field of <code>nl</code> is set to zero.				
RETURN VALUES	<code>kvm_nlist()</code> returns the value of <code>nlist(3UCB)</code> or <code>nlist(3ELF)</code> , depending on the library used.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>nlist(3UCB)</code> , <code>nlist(3ELF)</code> , <code>kvm_open(3KVM)</code> , <code>kvm_read(3KVM)</code> , <code>attributes(5)</code>				

kvm_open(3KVM)

NAME	kvm_open, kvm_close – specify a kernel to examine				
SYNOPSIS	<pre>#include <kvm.h> #include <fcntl.h> kvm_t *kvm_open(char *namelist, char *corefile, char *swapfile, int flag, char *errstr); int kvm_close(kvm_t *kd);</pre>				
DESCRIPTION	kvm_open() The <code>kvm_open()</code> function initializes a set of file descriptors to be used in subsequent calls to kernel virtual memory (VM) routines. It returns a pointer to a kernel identifier that must be used as the <i>kd</i> argument in subsequent kernel VM function calls. The <i>namelist</i> argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in <i>corefile</i>. If <i>namelist</i> is <code>NULL</code>, the symbol table of the currently running kernel is used to determine offsets in the core image. In this case, it is up to the implementation to select an appropriate way to resolve symbolic references, for instance, using <code>/dev/ksyms</code> as a default <i>namelist</i> file. The <i>corefile</i> argument specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see <code>savecore(1M)</code>) or the special device <code>/dev/mem</code>. If <i>corefile</i> is <code>NULL</code>, the currently running kernel is accessed, using <code>/dev/mem</code> and <code>/dev/kmem</code>. The <i>swapfile</i> argument specifies a file that represents the swap device. If both <i>corefile</i> and <i>swapfile</i> are <code>NULL</code>, the swap device of the currently running kernel is accessed. Otherwise, if <i>swapfile</i> is <code>NULL</code>, <code>kvm_open()</code> may succeed but subsequent <code>kvm_getu(3KVM)</code> function calls may fail if the desired information is swapped out. The <i>flag</i> function is used to specify read or write access for <i>corefile</i> and may have one of the following values: <table><tr><td><code>O_RDONLY</code></td><td>open for reading</td></tr><tr><td><code>O_RDWR</code></td><td>open for reading and writing</td></tr></table> The <i>errstr</i> argument is used to control error reporting. If it is a null pointer, no error messages will be printed. If it is non-null, it is assumed to be the address of a string that will be used to prefix error messages generated by <code>kvm_open</code>. Errors are printed to <code>stderr</code>. A useful value to supply for <i>errstr</i> would be <code>argv[0]</code>. This has the effect of printing the process name in front of any error messages. Applications using <code>libkvm</code> are dependent on the underlying data model of the kernel image, that is, whether it is a 32-bit or 64-bit kernel.	<code>O_RDONLY</code>	open for reading	<code>O_RDWR</code>	open for reading and writing
<code>O_RDONLY</code>	open for reading				
<code>O_RDWR</code>	open for reading and writing				

kvm_open(3KVM)

	The data model of these applications must match the data model of the kernel in order to correctly interpret the size and offsets of kernel data structures. For example, a 32-bit application that uses the 32-bit version of the <code>libkvm</code> interfaces will fail to open a 64-bit kernel image. Similarly, a 64-bit application that uses the 64-bit version of the <code>libkvm</code> interfaces will fail to open a 32-bit kernel image.				
kvm_close()	The <code>kvm_close()</code> function closes all file descriptors that were associated with <i>kd</i>. These files are also closed on <code>exit(2)</code> and <code>execve()</code> (see <code>exec(2)</code>). <code>kvm_close()</code> also resets the <code>proc</code> pointer associated with <code>kvm_nextproc(3KVM)</code> and flushes any cached kernel data.				
RETURN VALUES	The <code>kvm_open()</code> function returns a non-null value suitable for use with subsequent kernel VM function calls. On failure, it returns <code>NULL</code> and no files are opened. The <code>kvm_close()</code> function returns 0 on success -1 on failure.				
FILES	<code>/dev/kmem</code> <code>/dev/ksyms</code> <code>/dev/mem</code>				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>savecore(1M)</code>, <code>exec(2)</code>, <code>exit(2)</code>, <code>pathconf(2)</code>, <code>getloadavg(3C)</code>, <code>kstat(3KSTAT)</code>, <code>kvm_getu(3KVM)</code>, <code>kvm_nextproc(3KVM)</code>, <code>kvm_nlist(3KVM)</code>, <code>kvm_read(3KVM)</code>, <code>sysconf(3C)</code>, <code>libkvm(3LIB)</code>, <code>proc(4)</code>, <code>attributes(5)</code>, <code>lfcompile(5)</code>				
NOTES	Kernel core dumps should be examined on the platform on which they were created. While a 32-bit application running on a 64-bit kernel can examine a 32-bit core dump, a 64-bit application running on a 64-bit kernel cannot examine a kernel core dump from the 32-bit system. Applications using <code>libkvm</code> are likely to be platform- and release-dependent. On 32-bit systems, applications that use <code>libkvm</code> to access the running kernel must be 32-bit applications. On systems that support both 32-bit and 64-bit applications, applications that use the <code>libkvm</code> interfaces to access the running kernel must themselves be 64-bit applications. Most of the traditional uses of <code>libkvm</code> have been superseded by more stable interfaces that allow the same information to be extracted more efficiently, yet independent of the kernel data model. For examples, see <code>sysconf(3C)</code>, <code>proc(4)</code>, <code>kstat(3KSTAT)</code>, <code>getloadavg(3C)</code>, and <code>pathconf(2)</code>.				

kvm_read(3KVM)

NAME	kvm_read, kvm_write, kvm_uread, kvm_uwrite, kvm_kread, kvm_kwrite – copy data to or from a kernel image or running system
SYNOPSIS	<pre>#include <kvm.h> ssize_t kvm_read(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes); ssize_t kvm_write(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes); ssize_t kvm_kread(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes); ssize_t kvm_kwrite(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes); ssize_t kvm_uread(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes); ssize_t kvm_uwrite(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);</pre>
DESCRIPTION	
kvm_kread()	The <code>kvm_kread()</code> function transfers data from the kernel address space to the address space of the process. <i>nbytes</i> bytes of data are copied from the kernel virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> .
kvm_kwrite()	The <code>kvm_kwrite()</code> function is like <code>kvm_kread()</code> , except that the direction of the transfer is reversed. To use this function, the <code>kvm_open(3KVM)</code> call that returned <i>kd</i> must have specified write access.
kvm_uread()	The <code>kvm_uread()</code> function transfers data from the address space of the processes specified in the most recent <code>kvm_getu(3KVM)</code> call. <i>nbytes</i> bytes of data are copied from the user virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> .
kvm_uwrite()	The <code>kvm_uwrite()</code> function is like <code>kvm_uread()</code> , except that the direction of the transfer is reversed. To use this function, the <code>kvm_open(3KVM)</code> call that returned <i>kd</i> must have specified write access. The address is resolved in the address space of the process specified in the most recent <code>kvm_getu(3KVM)</code> call.
kvm_read()	The <code>kvm_read()</code> function transfers data from the kernel image specified by <i>kd</i> (see <code>kvm_open(3KVM)</code>) to the address space of the process. <i>nbytes</i> bytes of data are copied from the kernel virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> .
kvm_write()	The <code>kvm_write()</code> function is like <code>kvm_read()</code> , except that the direction of data transfer is reversed. To use this function, the <code>kvm_open(3KVM)</code> call that returned <i>kd</i> must have specified write access. If a user virtual address is given, it is resolved in the address space of the process specified in the most recent <code>kvm_getu(3KVM)</code> call.

kvm_read(3KVM)

USAGE The use of `kvm_read()` and `kvm_write()` is strongly discouraged. On some platforms, there is considerable ambiguity over which address space is to be accessed by these functions, possibly leading to unexpected results. The `kvm_kread()`, `kvm_kwrite()`, `kvm_uread()`, and `kvm_uwrite()` functions are much more clearly defined in this respect.

RETURN VALUES On success, these functions return the number of bytes actually transferred. On failure, they return `-1`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `kvm_getu(3KVM)`, `kvm_nlist(3KVM)`, `kvm_open(3KVM)`, `attributes(5)`

ld_support(3EXT)

NAME	ld_support, ld_atexit, ld_atexit64, ld_file, ld_file64, ld_input_done, ld_input_section, ld_input_section64, ld_section, ld_section64, ld_start, ld_start64, ld_version – link-editor support functions
SYNOPSIS	<pre>void ld_atexit(int status); void ld_atexit64(int status); void ld_file(const char *name, const Elf_Kind kind, int flags, Elf *elf); void ld_file64(const char *name, const Elf_Kind kind, int flags, Elf *elf); void ld_input_done(uint_t *flags); void ld_input_section(const char *name, Elf32_Shdr **shdr, Elf32_Word sndx, Elf_Data *data, Elf *elf, uint_t *flags); void ld_input_section64(const char *name, Elf64_Shdr **shdr, Elf64_Word sndx, Elf_Data *data, Elf *elf, uint_t *flags); void ld_section(const char *name, Elf32_Shdr shdr, Elf32_Word sndx, Elf_Data *data, Elf *elf); void ld_section64(const char *name, Elf64_Shdr shdr, Elf64_Word sndx, Elf_Data *data, Elf *elf); void ld_start(const char *name, const Elf32_Half type, const char *caller); void ld_start64(const char *name, const Elf64_Half type, const char *caller); void ld_version(uint_t version);</pre>
DESCRIPTION	A link-editor support library is a user-created shared object offering one or more of these interfaces that are called by the link-editor ld(1) at various stages of the link-editing process. See the <i>Linker and Libraries Guide</i> for a full description of the link-editor support mechanism.
SEE ALSO	ld(1) <i>Linker and Libraries Guide</i>

NAME	lgamma, lgamma_r, gamma, gamma_r – log gamma function
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> extern int signgam; double lgamma(double x); double gamma(double x); double lgamma_r(double x, int *signgamp); double gamma_r(double x, int *signgamp);</pre>
DESCRIPTION	The <code>lgamma()</code>, <code>gamma()</code>, <code>lgamma_r()</code>, and <code>gamma_r()</code> functions return $\ln \Gamma(x) $ where $\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$ for $x > 0$ and $\Gamma(x) = \pi / (\Gamma(1-x) \sin(\pi x))$ for $x < 1$. The <code>lgamma()</code> and <code>gamma()</code> functions use the external integer <code>signgam</code> to return the sign of $\sim(x)$ while <code>lgamma_r()</code> and <code>gamma_r()</code> use the user-allocated space addressed by <i>signgamp</i>.
IDIOSYNCRASIES	In the case of <code>lgamma()</code>, do <i>not</i> use the expression <code>signgam*exp(lgamma(x))</code> to compute $g := \Gamma(x)$

lgamma(3M)

Instead compute `lgamma()` first:

```
lg = lgamma(x); g = signgam*exp(lg);
```

only after `lgamma()` has returned can `signgam` be correct. Note that `|~(x)` must overflow when x is large enough, underflow when $-x$ is large enough, and generate a division by 0 exception at the singularities x a nonpositive integer.

RETURN VALUES

For exceptional cases, `matherr(3M)` tabulates the values to be returned as dictated by various Standards.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SEE ALSO

`matherr(3M)`, `attributes(5)`, `standards(5)`

NOTES

Although `lgamma_r()` is not mentioned by POSIX 1003.1c, it was added to complete the functionality provided by similar thread-safe functions.

The `gamma()` function is currently maintained for compatibility with SVID3. See `standards(5)`. The `gamma()` function and the `gamma_r()` function may be removed from a future release. The `lgamma()` and `lgamma_r()` functions should be used instead.

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

The `lgamma()` function is unsafe in multithreaded applications. The `lgamma_r()` function should be used instead.

NAME	<code>lgrp_affinity_get</code> , <code>lgrp_affinity_set</code> – get of set lgroup affinity
SYNOPSIS	<pre>cc [flag ...] file... -llgrp [library ...] #include <sys/lgrp_user.h> lgrp_affinity_t lgrp_affinity_get(idtype_t <i>idtype</i>, id_t <i>id</i>, lgrp_id_t <i>lgrp</i>); int lgrp_affinity_set(idtype_t <i>idtype</i>, id_t <i>id</i>, lgrp_id_t <i>lgrp</i>, lgrp_affinity_t <i>affinity</i>);</pre>
DESCRIPTION	The <code>lgrp_affinity_get()</code> function returns the affinity that the LWP or set of LWPs specified by the <i>idtype</i> and <i>id</i> arguments have for the given lgroup. The <code>lgrp_affinity_set()</code> function sets the affinity that the LWP or set of LWPs specified by <i>idtype</i> and <i>id</i> have for the given lgroup. The lgroup affinity can be set to <code>LGRP_AFF_STRONG</code>, <code>LGRP_AFF_WEAK</code>, or <code>LGRP_AFF_NONE</code>. If the <i>idtype</i> is <code>P_PID</code>, the affinity is retrieved for one of the LWPs in the process or set for all the LWPs of the process with process ID (PID) <i>id</i>. The affinity is retrieved or set for the LWP of the current process with LWP ID <i>id</i> if <i>idtype</i> is <code>P_LWPID</code>. If <i>id</i> is <code>P_MYID</code>, then the current LWP or process is specified. The operating system uses the lgroup affinities as advice on where to run a thread and allocate its memory and factors this advice in with other constraints. Processor binding and processor sets can restrict which lgroups a thread can run on, but do not change the lgroup affinities. Each thread can have an affinity for an lgroup in the system such that the thread will tend to be scheduled to run on that lgroup and allocate memory from there whenever possible. If the thread has affinity for more than one lgroup, the operating system will try to run the thread and allocate its memory on the lgroup for which it has the strongest affinity, then the next strongest, and so on up through some small, system-dependent number of these lgroup affinities. When multiple lgroups have the same affinity, the order of preference among them is unspecified and up to the operating system to choose. The lgroup with the strongest affinity that the thread can run on is known as its "home lgroup" (see <code>lgrp_home(3LGRP)</code>) and is usually the operating system's first choice of where to run the thread and allocate its memory. There are different levels of affinity that can be specified by a thread for a particular lgroup. The levels of affinity are the following from strongest to weakest: <pre>LGRP_AFF_STRONG /* strong affinity */ LGRP_AFF_WEAK /* weak affinity */ LGRP_AFF_NONE /* no affinity */</pre> The <code>LGRP_AFF_STRONG</code> affinity serves as a hint to the operating system that the calling thread has a strong affinity for the given lgroup. If this is the thread's home lgroup, the operating system will avoid rehoming it to another lgroup if possible. However, dynamic reconfiguration, processor offlining, processor binding, and processor set binding and manipulation are examples of events that can cause the operating system to change the thread's home lgroup for which it has a strong affinity.

lgrp_affinity_get(3LGRP)

	The LGRP_AFF_WEAK affinity is a hint to the operating system that the calling thread has a weak affinity for the given lgroup. If a thread has a weak affinity for its home lgroup, the operating system interprets this to mean that thread does not mind whether it is rehomed, unlike LGRP_AFF_STRONG. Load balancing, dynamic reconfiguration, processor binding, or processor set binding and manipulation are examples of events that can cause the operating system to change a thread's home lgroup for which it has a weak affinity. The LGRP_AFF_NONE affinity signifies no affinity and can be used to remove a thread's affinity for a particular lgroup. Initially, each thread has no affinity to any lgroup. If a thread has no lgroup affinities set, the operating system chooses a home lgroup for the thread with no affinity set.						
RETURN VALUES	Upon successful completion, lgrp_affinity_get() returns the affinity for the given lgroup. Upon successful completion, lgrp_affinity_set() return 0. Otherwise, both functions return -1 and set errno to indicate the error.						
ERRORS	The lgrp_affinity_get() and lgrp_affinity_set() functions will fail if: <table><tr><td>EINVAL</td><td>The specified lgroup, affinity, or ID type is not valid.</td></tr><tr><td>EPERM</td><td>The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the LWP(s).</td></tr><tr><td>ESRCH</td><td>The specified lgroup or LWP(s) was not found.</td></tr></table>	EINVAL	The specified lgroup, affinity, or ID type is not valid.	EPERM	The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the LWP(s).	ESRCH	The specified lgroup or LWP(s) was not found.
EINVAL	The specified lgroup, affinity, or ID type is not valid.						
EPERM	The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the LWP(s).						
ESRCH	The specified lgroup or LWP(s) was not found.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	lgrp_home(3LGRP), liblgrp(3LIB), attributes(5)						

NAME	lgrp_children – get children of given lgroup						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -llgrp [<i>library</i> ...] #include <sys/lgrp_user.h> int lgrp_children(lgrp_cookie_t <i>cookie</i>, lgrp_id_t <i>parent</i>, lgrp_id_t *<i>lgrp_array</i>, uint_t <i>lgrp_array_size</i>);</pre>						
DESCRIPTION	The <code>lgrp_children()</code> function takes a <i>cookie</i> representing a snapshot of the lgroup hierarchy retrieved from <code>lgrp_init(3LGRP)</code> and returns the number of lgroups that are children of the specified lgroup. If the <i>lgrp_array</i> and <i>lgrp_array_size</i> arguments are non-null, the array is filled with as many of the children lgroup IDs as will fit, given the size of the array.						
RETURN VALUES	– returns the number of child lgroup IDs. Otherwise, it returns –1 and sets <code>errno</code> to indicate the error.						
ERRORS	The <code>lgrp_children()</code> function will fail if: <table> <tr> <td>EINVAL</td><td>The specified lgroup ID is not valid or the cookie is invalid.</td></tr> <tr> <td>ESRCH</td><td>The specified lgroup ID was not found.</td></tr> </table>	EINVAL	The specified lgroup ID is not valid or the cookie is invalid.	ESRCH	The specified lgroup ID was not found.		
EINVAL	The specified lgroup ID is not valid or the cookie is invalid.						
ESRCH	The specified lgroup ID was not found.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code>, <code>lgrp_nlgrps(3LGRP)</code>, <code>lgrp_parents(3LGRP)</code>, <code>liblgrp(3LIB)</code>, <code>attributes(5)</code>						

lgrp_cookie_stale(3LGRP)

NAME	lgrp_cookie_stale – determine whether snapshot of lgroup hierarchy is stale						
SYNOPSIS	<pre>cc [flag ...] file... -llgrp [library ...] #include <sys/lgrp_user.h> int lgrp_cookie_stale(lgrp_cookie_t cookie);</pre>						
DESCRIPTION	The <code>lgrp_cookie_stale()</code> function takes a <i>cookie</i> representing the snapshot of the lgroup hierarchy obtained from <code>lgrp_init(3LGRP)</code> and returns whether it is stale. The snapshot can become out-of-date for a number of reasons depending on its view. If the snapshot was taken with <code>LGRP_VIEW_OS</code>, changes in the lgroup hierarchy from dynamic reconfiguration, CPU on/offline, or other conditions can cause the snapshot to become out-of-date. A snapshot taken with <code>LGRP_VIEW_CALLER</code> can be affected by the caller's processor set binding and changes in its processor set itself, as well as changes in the lgroup hierarchy. If the snapshot needs to be updated, <code>lgrp_fini(3LGRP)</code> should be called with the old cookie and <code>lgrp_init()</code> should be called to obtain a new snapshot.						
RETURN VALUES	Upon successful completion, <code>lgrp_cookie_stale()</code> returns whether the cookie is stale. Otherwise, it returns <code>-1</code> and sets <code>errno</code> to indicate the error.						
ERRORS	The <code>lgrp_cookie_stale()</code> function will fail if: EINVAL The cookie is not valid.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>lgrp_fini(3LGRP)</code> , <code>lgrp_view(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	<code>lgrp_cpus</code> – get CPU IDs contained in specified lgroup						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llgrp [<i>library...</i>] #include <sys/lgrp_user.h> int lgrp_cpus(lgrp_cookie_t <i>cookie</i>, lgrp_id_t <i>lgrp</i>, processorid_t *<i>cpuids</i>, uint_t <i>count</i>, int <i>content</i>);</pre>						
DESCRIPTION	The <code>lgrp_cpus()</code> function takes a <i>cookie</i> representing a snapshot of the lgroup hierarchy obtained from <code>lgrp_init(3LGRP)</code> and returns the number of CPUs in the lgroup specified by <i>lgrp</i>. If both the <i>cpuids[]</i> argument is non-null and the count is non-zero, <code>lgrp_cpus()</code> stores up to the specified count of CPU IDs into the <i>cpuids[]</i> array. The <i>content</i> argument should be set to one of the following values to specify whether the direct contents or everything in this lgroup including its children should be returned: <pre>LGRP_CONTENT_HIERARCHY /* everything within this hierarchy */ LGRP_CONTENT_DIRECT /* directly contained in lgroup */</pre>						
RETURN VALUES	Upon successful completion, the number of CPUs in the given lgroup is returned. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_cpus()</code> function will fail if: <code>EINVAL</code> The specified cookie, lgroup ID, or one of the flags is not valid. <code>ESRCH</code> The specified lgroup ID was not found.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>lgrp_mem_size(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

lgrp_fini(3LGRP)

NAME	<code>lgrp_fini</code> – finished using lgroup interface						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llgrp [<i>library...</i>] #include <sys/lgrp_user.h> int lgrp_fini(lgrp_cookie_t <i>cookie</i>);</pre>						
DESCRIPTION	The <code>lgrp_fini()</code> function takes a <i>cookie</i> , frees the snapshot of the lgroup hierarchy created by <code>lgrp_init(3LGRP)</code> , and cleans up anything else set up by <code>lgrp_init()</code> . After this function is called, any memory allocated and returned by the lgroup interface might no longer be valid and should not be used.						
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_fini()</code> function will fail if: EINVAL The <i>cookie</i> is not valid.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>lgrp_cookie_stale(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	<code>lgrp_home</code> – get home lgroup						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llgrp [<i>library...</i>] #include <sys/lgrp_user.h> lgrp_id_t lgrp_home(idtype_t <i>idtype</i>, id_t <i>id</i>);</pre>						
DESCRIPTION	The <code>lgrp_home()</code> function returns the ID of the home lgroup for the given process or thread. A thread can have an affinity for an lgroup in the system such that the thread will tend to be scheduled to run on that lgroup and allocate memory from there whenever possible. The lgroup with the strongest affinity that the thread can run on is known as the "home lgroup" of the thread. If the thread has no affinity for any lgroup that it can run on, the operating system will choose a home for it. The <i>idtype</i> argument should be <code>P_PID</code> to specify a process and the <i>id</i> argument should be its process ID. Otherwise, the <i>idtype</i> argument should be <code>P_LWPID</code> to specify a thread and the <i>id</i> argument should be its LWP ID. The value <code>P_MYID</code> can be used for the <i>id</i> argument to specify the current process or thread.						
RETURN VALUES	Upon successful completion, <code>lgrp_home()</code> returns the ID of the home lgroup of the specified process or thread. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_home()</code> function will fail if: <table> <tr> <td><code>EINVAL</code></td><td>The ID type is not valid.</td></tr> <tr> <td><code>EPERM</code></td><td>The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the threads.</td></tr> <tr> <td><code>ESRCH</code></td><td>The specified process or thread was not found.</td></tr> </table>	<code>EINVAL</code>	The ID type is not valid.	<code>EPERM</code>	The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the threads.	<code>ESRCH</code>	The specified process or thread was not found.
<code>EINVAL</code>	The ID type is not valid.						
<code>EPERM</code>	The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the threads.						
<code>ESRCH</code>	The specified process or thread was not found.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_affinity_get(3LGRP)</code> , <code>lgrp_init(3LGRP)</code> , <code>attributes(5)</code>						

lgrp_init(3LGRP)

NAME	<code>lgrp_init</code> – initialize lgroup interface						
SYNOPSIS	<pre>cc [flag ...] file... -llgrp [library ...] #include <sys/lgrp_user.h> lgrp_cookie_t lgrp_init(lgrp_view_t view);</pre>						
DESCRIPTION	The <code>lgrp_init()</code> function initializes the lgroup interface and takes a snapshot of the lgroup hierarchy with the given <i>view</i>. If the given <i>view</i> is <code>LGRP_VIEW_CALLER</code>, the snapshot contains only the resources that are available to the caller (for example, with respect to processor sets). When the <i>view</i> is <code>LGRP_VIEW_OS</code>, the snapshot contains what is available to the operating system. Given the <i>view</i>, <code>lgrp_init()</code> returns a cookie representing this snapshot of the lgroup hierarchy. This cookie should be used with other routines in the lgroup interface needing the lgroup hierarchy. The <code>lgrp_fini(3LGRP)</code> function should be called with the cookie when it is no longer needed. The lgroup hierarchy consists of a root lgroup, which is the maximum bounding locality group of the system. It contains all the CPU and memory resources of the machine and can contain other locality groups that contain CPUs and memory within a smaller locality.						
RETURN VALUES	Upon successful completion, <code>lgrp_init()</code> returns a cookie. Otherwise it returns <code>LGRP_COOKIE_NONE</code> and sets <code>errno</code> to indicate the error.						
ERRORS	The <code>lgrp_init()</code> function will fail if: <table><tr><td><code>EINVAL</code></td><td>The view is not valid.</td></tr><tr><td><code>ENOMEM</code></td><td>There was not enough memory to allocate the snapshot of the lgroup hierarchy.</td></tr></table>	<code>EINVAL</code>	The view is not valid.	<code>ENOMEM</code>	There was not enough memory to allocate the snapshot of the lgroup hierarchy.		
<code>EINVAL</code>	The view is not valid.						
<code>ENOMEM</code>	There was not enough memory to allocate the snapshot of the lgroup hierarchy.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_children(3LGRP)</code> , <code>lgrp_cookie_stale(3LGRP)</code> , <code>lgrp_cpus(3LGRP)</code> , <code>lgrp_fini(3LGRP)</code> , <code>lgrp_mem_size(3LGRP)</code> , <code>lgrp_nlgrps(3LGRP)</code> , <code>lgrp_parents(3LGRP)</code> , <code>lgrp_root(3LGRP)</code> , <code>lgrp_view(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	lgrp_latency – get latency between two lgroups						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llgrp [<i>library...</i>] #include <sys/lgrp_user.h> int lgrp_latency(lgrp_id_t <i>from</i>, lgrp_id_t <i>to</i>);</pre>						
DESCRIPTION	The <code>lgrp_latency()</code> function returns the latency value between a CPU in the <i>from</i> lgroup to the memory in the <i>to</i> lgroup. If <i>from</i> is the same lgroup as <i>to</i>, the latency value within that lgroup is returned. The latency value is defined by the operating system and is platform-specific. It can be used only for relative comparison of lgroups on the running system. It does not necessarily represent the actual latency between hardware devices, and it might not be applicable across platforms.						
RETURN VALUES	Upon successful completion, the latency value is returned. Otherwise –1 is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_latency()</code> function will fail if: <table> <tr> <td>EINVAL</td><td>The specified lgroup ID is not valid.</td></tr> <tr> <td>ESRCH</td><td>The specified lgroup ID was not found, the <i>from</i> lgroup does not contain any CPUs, or the <i>to</i> lgroup does not have any memory.</td></tr> </table>	EINVAL	The specified lgroup ID is not valid.	ESRCH	The specified lgroup ID was not found, the <i>from</i> lgroup does not contain any CPUs, or the <i>to</i> lgroup does not have any memory.		
EINVAL	The specified lgroup ID is not valid.						
ESRCH	The specified lgroup ID was not found, the <i>from</i> lgroup does not contain any CPUs, or the <i>to</i> lgroup does not have any memory.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>lgrp_parents(3LGRP)</code> , <code>lgrp_children(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

lgrp_mem_size(3LGRP)

NAME	lgrp_mem_size – return the memory size of the given lgroup						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llgrp [<i>library...</i>] #include <sys/lgrp_user.h> lgrp_mem_size_t lgrp_mem_size(lgrp_cookie_t <i>cookie</i>, lgrp_id_t <i>lgrp</i>, int <i>type</i>, int <i>content</i>);</pre>						
DESCRIPTION	The <code>lgrp_mem_size()</code> function takes a <i>cookie</i> representing a snapshot of the lgroup hierarchy. The <i>cookie</i> was obtained by calling <code>lgrp_init(3LGRP)</code>. The <code>lgrp_mem_size()</code> function returns the memory size of the given lgroup in bytes. The <i>type</i> argument should be set to one of the following values: <pre>LGRP_MEM_SZ_FREE /* free memory */ LGRP_MEM_SZ_INSTALLED /* installed memory */</pre> The <i>content</i> argument should be set to one of the following values to specify whether the direct contents or everything in this lgroup including its children should be returned: <pre>LGRP_CONTENT_HIERARCHY /* everything within this hierarchy */ LGRP_CONTENT_DIRECT /* directly contained in lgroup */</pre> The total sizes include all the memory in the lgroup including its children, while the others reflect only the memory contained directly in the given lgroup.						
RETURN VALUES	Upon successful completion, the size in bytes is returned. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_mem_size()</code> function will fail if: <table><tr><td>EINVAL</td><td>The specified cookie, lgroup ID, or one of the flags is not valid.</td></tr><tr><td>ESRCH</td><td>The specified lgroup ID was not found.</td></tr></table>	EINVAL	The specified cookie, lgroup ID, or one of the flags is not valid.	ESRCH	The specified lgroup ID was not found.		
EINVAL	The specified cookie, lgroup ID, or one of the flags is not valid.						
ESRCH	The specified lgroup ID was not found.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>lgrp_cpus(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	<code>lgrp_nlgrps</code> – get number of lgroups
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -llgrp [<i>library</i> ...] #include <sys/lgrp_user.h> int lgrp_nlgrps(lgrp_cookie_t <i>cookie</i>);</pre>
DESCRIPTION	The <code>lgrp_nlgrps()</code> function takes a <i>cookie</i> representing a snapshot of the lgroup hierarchy obtained from <code>lgrp_init(3LGRP)</code> . It returns the number of lgroups in the hierarchy where the number is always at least one.
RETURN VALUES	Upon successful completion, <code>lgrp_nlgrps()</code> returns the number of lgroups in the system. Otherwise, it returns <code>-1</code> and sets <code>errno</code> to indicate the error.
ERRORS	The <code>lgrp_nlgrps()</code> function will fail if: EINVAL The <i>cookie</i> is not valid.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

lgrp_parents(3LGRP)

NAME	<code>lgrp_parents</code> – get parents of given lgroup						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -llgrp [<i>library</i> ...] #include <sys/lgrp_user.h> int lgrp_parents(lgrp_cookie_t <i>cookie</i>, lgrp_id_t <i>child</i>, lgrp_id_t *<i>lgrp_array</i>, uint_t <i>lgrp_array_size</i>);</pre>						
DESCRIPTION	The <code>lgrp_parents()</code> function takes a <i>cookie</i> representing a snapshot of the lgroup hierarchy obtained from <code>lgrp_init(3LGRP)</code> and returns the number of parent lgroups of the specified lgroup. If <i>lgrp_array</i> is non-null and the <i>lgrp_array_size</i> is non-zero, the array is filled with as many of the parent lgroup IDs as will fit given the size of the array. For the root lgroup, the number of parents returned is 0 and the <i>lgrp_array</i> argument is not filled in.						
RETURN VALUES	Upon successful completion, <code>lgrp_parents()</code> returns the number of parent lgroup IDs. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_parents()</code> function will fail if: EINVAL The specified cookie or lgroup ID is not valid. ESRCH The specified lgroup ID was not found.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_children(3LGRP)</code> , <code>lgrp_init(3LGRP)</code> , <code>lgrp_nlgrps(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	<code>lgrp_root</code> – return root lgroup ID
SYNOPSIS	<pre>cc [<i>flag</i>...] <i>file</i>... -llgrp [<i>library</i>...] #include <sys/lgrp_user.h> lgrp_id_t lgrp_root(lgrp_cookie_t <i>cookie</i>) ;</pre>
DESCRIPTION	The <code>lgrp_root()</code> function returns the root lgroup ID.
RETURN VALUES	Upon successful completion, <code>lgrp_root()</code> returns the lgroup ID of the root lgroup.
ERRORS	The <code>lgrp_root()</code> function will fail if: <code>EINVAL</code> The <i>cookie</i> is not valid.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_nlgrps(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

lgrp_version(3LGRP)

NAME	lgrp_version – coordinate library and application versions						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -llgrp [<i>library</i> ...] #include <sys/lgrp_user.h> int lgrp_version(const int <i>version</i>);</pre>						
DESCRIPTION	The <code>lgrp_version()</code> function takes an interface version number, <i>version</i> , as an argument and returns an lgroup interface version. The <i>version</i> argument should be the value of <code>LGRP_VER_CURRENT</code> bound to the application when it was compiled or <code>LGRP_VER_NONE</code> to find out the current lgroup interface version on the running system.						
RETURN VALUES	If <i>version</i> is still supported by the implementation, then <code>lgrp_version()</code> returns the requested version. If <code>LGRP_VER_NONE</code> is returned, the implementation cannot support the requested version. The application should be recompiled and might require further changes. If <i>version</i> is <code>LGRP_VER_NONE</code>, <code>lgrp_version()</code> returns the current version of the library.						
EXAMPLES	EXAMPLE 1 Test whether the version of the interface used by the caller is supported. The following example tests whether the version of the interface used by the caller is supported: <pre>#include <sys/lgrp_user.h> if (lgrp_version(LGRP_VER_CURRENT) != LGRP_VER_CURRENT) { fprintf(stderr, "Built with unsupported lgroup interface %d\n", LGRP_VER_CURRENT); exit (1); }</pre>						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_init(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

NAME	lgrp_view – get view of lgroup hierarchy						
SYNOPSIS	<pre>cc [flag...] file... -llgrp [library...] #include <sys/lgrp_user.h> lgrp_view_t lgrp_view(lgrp_cookie_t cookie);</pre>						
DESCRIPTION	The <code>lgrp_view()</code> function takes a <i>cookie</i> representing the snapshot of the lgroup hierarchy obtained from <code>lgrp_init(3LGRP)</code> and returns the snapshot's view of the lgroup hierarchy. If the given view is <code>LGRP_VIEW_CALLER</code>, the snapshot contains only the resources that are available to the caller (such as those with respect to processor sets). When the view is <code>LGRP_VIEW_OS</code>, the snapshot contains what is available to the operating system.						
RETURN VALUES	Upon succesful completion, <code>lgrp_view()</code> returns the view for the snapshot of the lgroup hierarchy represented by the given cookie. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.						
ERRORS	The <code>lgrp_view()</code> function will fail if: <code>EINVAL</code> The <i>cookie</i> is not valid.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>lgrp_cookie_stale(3LGRP)</code> , <code>lgrp_fini(3LGRP)</code> , <code>lgrp_init(3LGRP)</code> , <code>liblgrp(3LIB)</code> , <code>attributes(5)</code>						

libdevinfo(3DEVINFO)

NAME	libdevinfo – library of device information functions
SYNOPSIS	<pre>cc [flag ...] file ...-ldevinfo [library ...] #include <libdevinfo.h></pre>
DESCRIPTION	The libdevinfo library contains a set of interfaces for accessing device configuration data. Device configuration data is organized as a tree of device nodes, defined as <code>di_node_t</code> in the libdevinfo interfaces. Each <code>di_node_t</code> represents a physical or logical (pseudo) device. Three types of data are associated with device nodes: ■ data defined for all device nodes (attributes) ■ properties specific to each device ■ minor node data All device nodes have a set of common attributes, such as a node name, an instance number, and a driver binding name. Common device node attributes are accessed by calling interfaces listed on the <code>di_binding_name(3DEVINFO)</code> man page. Each device node also has a physical path, which is accessed by calling <code>di_devfs_path(3DEVINFO)</code>. Properties provide device specific information for device configuration and usage. Properties may be defined by software (<code>di_prop_t</code>) or by firmware (<code>di_prom_prop_t</code>). One way to access each <code>di_prop_t</code> is to make successive calls to <code>di_prop_next(3DEVINFO)</code> until <code>DI_PROP_NIL</code> is returned. For each <code>di_prop_t</code>, use interfaces on the <code>di_prop_bytes(3DEVINFO)</code> man page to obtain property names and values. Another way to access these properties is to call <code>di_prop_lookup_bytes(3DEVINFO)</code> to find the value of a property with a given name. Accessing a <code>di_prom_prop_t</code> is similar to accessing a <code>di_prop_t</code>, except that the interface names start with <code>di_prom_prop</code> and additional calls to <code>di_prom_init(3DEVINFO)</code> and <code>di_prom_fini(3DEVINFO)</code> are required. Minor nodes contain information exported by the device for creating special files for the device. Each device node has 0 or more minor nodes associated with it. A list of minor nodes (<code>di_minor_t</code>) may be obtained by making successive calls to <code>di_minor_next(3DEVINFO)</code> until <code>DI_MINOR_NIL</code> is returned. For each minor node, <code>di_minor_devt(3DEVINFO)</code> and related interfaces are called to get minor node data. Using libdevinfo involves three steps: ■ Creating a snapshot of the device tree ■ Traversing the device tree to get information of interest ■ Destroying the snapshot of the device tree

A snapshot of the device tree is created by calling `di_init(3DEVINFO)` and destroyed by calling `di_fini(3DEVINFO)`. An application can specify the data to be included in the snapshot (full or partial tree, include or exclude properties and minor nodes) and get a handle to the root of the device tree. See `di_init(3DEVINFO)` for details. The application then traverses the device tree in the snapshot to obtain device configuration data.

The device tree is normally traversed through parent-child-sibling linkage. Each device node contains references to its parent, its next sibling, and the first of its children. Given the `di_node_t` returned from `di_init(3DEVINFO)`, one can find all children by first calling `di_child_node(3DEVINFO)`, followed by successive calls to `di_sibling_node(3DEVINFO)`, until `DI_NODE_NIL` is returned. By following this procedure recursively, an application can visit all device nodes contained in the snapshot. Two interfaces, `di_walk_node(3DEVINFO)` and `di_walk_minor(3DEVINFO)`, are provided to facilitate device tree traversal. The `di_walk_node(3DEVINFO)` interface visits all device nodes and executes a user-supplied callback function for each node visited. The `di_walk_minor(3DEVINFO)` does the same for each minor node in the device tree.

An alternative way to traverse the device tree is through the per-driver device node linkage. Device nodes contain a reference to the next device node bound to the same driver. Given the `di_node_t` returned from `di_init(3DEVINFO)`, an application can find all device nodes bound to a driver by first calling `di_drv_first_node(3DEVINFO)`, followed by successive calls to `di_drv_next_node(3DEVINFO)` until `DI_NODE_NIL` is returned. Note that traversing the per-driver device node list works only when the snapshot includes all device nodes.

See `libdevinfo(3LIB)` for a complete list of `libdevinfo` interfaces. See `di_init(3DEVINFO)` for examples of `libdevinfo` usage. See *Writing Device Drivers* for details of Solaris device configuration.

EXAMPLES

EXAMPLE 1 Information Accessible Through libdevinfo Interfaces

The following example illustrates the kind of information accessible through `libdevinfo` interfaces for a device node representing a hard disk (sd2):

```
Attributes
  node name:  sd
  instance:   2
  physical path:  /sbus@1f,0/espdma@e,8400000/esp@e,8800000/sd@2,0

Properties
  target=2
  lun=0

Minor nodes
  (disk partition /dev/dsk/c0t2d0s0)
    name:      a
    dev_t:     0x0080010 (32/16)
    spectype:  IF_BLK (block special)
  (disk partition /dev/rdisk/c0t2d0s2)
```

EXAMPLE 1 Information Accessible Through libdevinfo Interfaces (Continued)

```

name:      c,raw
dev_t:     0x0080012 (32/18)
spectype:  IF_CHR (character special)

```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe
Interface Stability	Evolving

SEE ALSO

devlinks(1M), prtconf(1M), di_binding_name(3DEVINFO),
 di_child_node(3DEVINFO), di_devfs_path(3DEVINFO),
 di_drv_first_node(3DEVINFO), di_drv_next_node(3DEVINFO),
 di_fini(3DEVINFO), di_init(3DEVINFO), di_minor_devt(3DEVINFO),
 di_minor_next(3DEVINFO), di_prom_fini(3DEVINFO),
 di_prom_init(3DEVINFO), di_prop_bytes(3DEVINFO),
 di_prop_lookup_bytes(3DEVINFO), di_prop_next(3DEVINFO),
 di_sibling_node(3DEVINFO), di_walk_minor(3DEVINFO),
 di_walk_node(3DEVINFO), libdevinfo(3LIB), attributes(5)

Writing Device Drivers

NAME	libnvpair – library of name-value pair functions						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lnvpair [<i>library...</i>] #include <libnvpair.h></pre>						
DESCRIPTION	The libnvpair library exports a set of functions for managing name-value pairs. The library defines two opaque handles: nvpair_t handle to a name-value pair nvlist_t handle to a list of name-value pairs The library supports the following operations: ■ Allocate and free an nvlist_t. ■ Add and remove an nvpair_t from a list. ■ Search nvlist_t for a specified name pair. ■ Pack an nvlist_t into a contiguous buffer. ■ Expand a packed nvlist into a searchable nvlist_t. See libnvpair(3LIB) for a complete list of libnvpair functions.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	libnvpair(3LIB), attributes(5)						

libpicl(3PICL)

NAME	libpicl – PICL interface library												
SYNOPSIS	<pre>cc [<i>flag</i> . . .] <i>file</i> . . . -lpicl [<i>library</i> . . .] #include <picl.h></pre>												
DESCRIPTION	The PICL interface is the platform-independent interface for clients to access the platform information. The set of functions and data structures of this interface are defined in the <code><picl.h></code> header. The information published through PICL is organized in a tree, where each node is an instance of a well-defined PICL class. The functions in the PICL interface allow the clients to access the properties of the nodes. The name of the base PICL class is <code>picl</code>, which defines a basic set of properties that all nodes in the tree must possess. The following table shows the property set of a <code>picl</code> class node. <table><tr><th>Property Name</th><th>Property Value</th></tr><tr><td><code>name</code></td><td>The name of the node</td></tr><tr><td><code>_class</code></td><td>The PICL class name of the node</td></tr><tr><td><code>_parent</code></td><td>Node handle of the parent node</td></tr><tr><td><code>_child</code></td><td>Node handle of the first child node</td></tr><tr><td><code>_peer</code></td><td>Node handle of the next peer node</td></tr></table> Property names with a leading underscore ('_') are reserved for use by the PICL framework. The property names <code>_class</code>, <code>_parent</code>, <code>_child</code>, and <code>_peer</code> are reserved names of the PICL framework, and are used to refer to a node's parent, child, and peer nodes, respectively. A client shall access a reserved property by their names only as they do not have an associated handle. The property name is not a reserved property, but a mandatory property for all nodes. Properties are classified into different types. Properties of type integer, unsigned-integer, and float have integer, unsigned integer, and floating-point values, respectively. A <code>table</code> property type has the handle to a table as its value. A table is a matrix of properties. A <code>reference</code> property type has a handle to a node in the tree as its value. A <code>reference</code> property may be used to establish an association between any two nodes in the tree. A <code>timestamp</code> property type has the value of time in seconds since Epoch. A <code>bytearray</code> property type has an array of bytes as its value. A <code>charstring</code> property type has a nul ('\0') terminated sequence of ASCII characters. The size of a property specifies the size of its value in bytes. A <code>void</code> property type denotes a property that exists but has no value. The following table lists the different PICL property types enumerated in <code>picl_prop_type_t</code>.	Property Name	Property Value	<code>name</code>	The name of the node	<code>_class</code>	The PICL class name of the node	<code>_parent</code>	Node handle of the parent node	<code>_child</code>	Node handle of the first child node	<code>_peer</code>	Node handle of the next peer node
Property Name	Property Value												
<code>name</code>	The name of the node												
<code>_class</code>	The PICL class name of the node												
<code>_parent</code>	Node handle of the parent node												
<code>_child</code>	Node handle of the first child node												
<code>_peer</code>	Node handle of the next peer node												

Property Type	Property Value
PICL_PTYPE_VOID	None
PICL_PTYPE_INT	Is an integer
PICL_PTYPE_UNSIGNED_INT	Is an unsigned integer
PICL_PTYPE_FLOAT	Is a floating-point number
PICL_PTYPE_REFERENCE	Is a PICL node handle

Reference Property Naming Convention

Reference properties may be used by plug-ins to publish properties in nodes of different classes. To make these property names unique, their names must be prefixed by `_picl_class_name_`, where *picl_class_name* is the class name of the node referenced by the property. Valid PICL class names are combinations of uppercase and lowercase letters 'a' through 'z', digits '0' through '9', and '-' (minus) characters. The string that follows the '*_picl_class_name_*' portion of a reference property name may be used to indicate a specific property in the referenced class, when applicable.

Property Information

The information about a node's property that can be accessed by PICL clients is defined by the `picl_propinfo_t` structure.

```
typedef struct {
    picl_prop_type_t  type;           /* property type */
    unsigned int      accessmode;     /* read, write */
    size_t            size;           /* item size or string size */
    char              name[PICL_PROPNAMELEN_MAX];
} picl_propinfo_t;
```

The `type` member specifies the property value type and the `accessmode` specifies the allowable access to the property. The plug-in module that adds the property to the PICL tree also sets the access mode of that property. The volatile nature of a property created by the plug-in is not visible to the PICL clients. The `size` member specifies the number of bytes occupied by the property's value. The maximum allowable size of property value is `PICL_PROPSIZE_MAX`, which is set to 512KB.

Property Access Modes

The plug-in module may publish a property granting a combination of the following access modes to the clients:

```
#define PICL_READ    0x1    /* read permission */
#define PICL_WRITE   0x2    /* write permission */
```

Property Names

The maximum length of the name of any property is specified by `PICL_PROPNAMELEN_MAX`.

Class Names

The maximum length of a PICL class name is specified by `PICL_CLASSNAMELEN_MAX`.

libpicl(3PICL)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `libpicl(3LIB)`, `attributes(5)`

NAME	libpicltree – PTree and Plug-in Registration interface library
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h></pre>
DESCRIPTION	The PTree interface is the set of functions and data structures to access and manipulate the PICL tree. The daemon and the plug-in modules use the PTree interface. The Plug-in Registration interface is used by the plug-in modules to register themselves with the daemon. The plug-in modules create the nodes and properties of the tree. At the time of creating a property, the plug-ins specify the property information in the <code>ptree_propinfo_t</code> structure defined as: <pre>typedef struct { int version; /* version */ picl_propinfo_t piclinfo; /* info to clients */ int (*read)(ptree_rarg_t *arg, void *buf); /* read access function for */ /* volatile prop */ int (*write)(ptree_warg_t *arg, const void *buf); /* write access function for */ /* volatile prop */ } ptree_propinfo_t;</pre> See <code>libpicl(3PICL)</code> for more information on PICL tree nodes and properties. The maximum size of a property value cannot exceed <code>PICL_PROPSIZE_MAX</code>. It is currently set to 512KB. Volatile Properties In addition to <code>PICL_READ</code> and <code>PICL_WRITE</code> property access modes, the plug-in modules specify whether a property is volatile or not by setting the bit <code>PICL_VOLATILE</code>. <pre>#define PICL_VOLATILE 0x4</pre> For a volatile property, the plug-in module provides the access functions to read and/or write the property in the <code>ptree_propinfo_t</code> argument passed when creating the property. The daemon invokes the access functions of volatile properties when clients access their values. Two arguments are passed to the read access functions. The first argument is a pointer to <code>ptree_rarg_t</code>, which contains the handle of the node, the handle of the accessed property and the credentials of the caller. The second argument is a pointer to the buffer where the value is to be copied. <pre>typedef struct { picl_nodehdl_t nodeh; picl_prophdl_t proph; door_cred_t cred; } ptree_rarg_t;</pre> The prototype of the read access function for volatile property is:

libpicltree(3PICLTREE)

```
int read(ptree_rarg_t *rarg, void *buf);
```

The read function returns `PICL_SUCCESS` to indicate successful completion.

Similarly, when a write access is performed on a volatile property, the daemon invokes the write access function provided by the plug-in for that property and passes it two arguments. The first argument is a pointer to `ptree_warg_t`, which contains the handle to the node, the handle of the accessed property and the credentials of the caller. The second argument is a pointer to the buffer containing the value to be written.

```
typedef struct {
    picl_nodehdl_t  nodeh;
    picl_prophdl_t  proph;
    door_cred_t     cred;
} ptree_warg_t;
```

The prototype of the write access function for volatile property is:

```
int write(ptree_warg_t *warg, const void *buf);
```

The write function returns `PICL_SUCCESS` to indicate successful completion.

For all volatile properties, the 'size' of the property must be specified to be the maximum possible size of the value. The maximum size of the value cannot exceed `PICL_PROPSIZE_MAX`. This allows a client to allocate a sufficiently large buffer before retrieving a volatile property's value

Plug-in Modules

Plug-in modules are shared objects that are located in well-known directories for the daemon to locate and load them. Plug-in module's are located in the one of the following plug-in directories depending on the platform-specific nature of the data they collect and publish.

```
/usr/platform/picl/plugins/'uname -i'/
/usr/platform/picl/plugins/'uname -m'/
/usr/lib/picl/plugins/
```

A plug-in module may specify its dependency on another plug-in module using the `-l` linker option. The plug-ins are loaded by the PICL daemon using `dlopen(3DL)` according to the specified dependencies. Each plug-in module must define a `.init` section, which is executed when the plug-in module is loaded, to register themselves with the daemon. See `picld_plugin_register(3PICLTREE)` for more information on plug-in registration.

The plug-in modules may use the `picld_log(3PICLTREE)` function to log their messages to the system log file.

ATTRIBUTES

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

libpicltree(3PICLTREE)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO libpicl(3PICL), libpicltree(3LIB), picld_log(3PICLTREE),
picld_plugin_register(3PICLTREE), attributes(5)

libtnfctl(3TNF)

NAME	libtnfctl – library for TNF probe control in a process or the kernel														
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h></pre>														
DESCRIPTION	The libtnfctl library provides an API to control TNF ("Trace Normal Form") probes within a process or the kernel. See tracing(3TNF) for an overview of the Solaris tracing architecture. The client of libtnfctl controls probes in one of four modes: <table><tr><td>internal mode</td><td>The target is the controlling process itself; that is, the client controls its own probes.</td></tr><tr><td>direct mode</td><td>The target is a separate process; a client can either exec(2) a program or attach to a running process for probe control. The libtnfctl library uses proc(4) on the target process for probe and process control in this mode, and additionally provides basic process control features.</td></tr><tr><td>indirect mode</td><td>The target is a separate process, but the controlling process is already using proc(4) to control the target, and hence libtnfctl cannot use those interfaces directly. Use this mode to control probes from within a debugger. In this mode, the client must provide a set of functions that libtnfctl can use to query and update the target process.</td></tr><tr><td>kernel mode</td><td>The target is the Solaris kernel.</td></tr></table> A process is controlled "externally" if it is being controlled in either direct mode or indirect mode. Alternatively, a process is controlled "internally" when it uses internal mode to control its own probes. There can be only one client at a time doing probe control on a given process. Therefore, it is not possible for a process to be controlled internally while it is being controlled externally. It is also not possible to have a process controlled by multiple external processes. Similarly, there can be only one process at a time doing kernel probe control. Note, however, that while a given target may only be controlled by one libtnfctl client, a single client may control an arbitrary number of targets. That is, it is possible for a process to simultaneously control its own probes, probes in other processes, and probes in the kernel. The following tables denotes the modes applicable to all libtnfctl interfaces (INT = internal mode; D = direct mode; IND = indirect mode; K = kernel mode). These interfaces create handles in the specified modes: <table><tr><td>tnfctl_internal_open()</td><td>INT</td></tr><tr><td>tnfctl_exec_open()</td><td>D</td></tr><tr><td>tnfctl_pid_open()</td><td>D</td></tr></table>	internal mode	The target is the controlling process itself; that is, the client controls its own probes.	direct mode	The target is a separate process; a client can either exec(2) a program or attach to a running process for probe control. The libtnfctl library uses proc(4) on the target process for probe and process control in this mode, and additionally provides basic process control features.	indirect mode	The target is a separate process, but the controlling process is already using proc(4) to control the target, and hence libtnfctl cannot use those interfaces directly. Use this mode to control probes from within a debugger. In this mode, the client must provide a set of functions that libtnfctl can use to query and update the target process.	kernel mode	The target is the Solaris kernel.	tnfctl_internal_open()	INT	tnfctl_exec_open()	D	tnfctl_pid_open()	D
internal mode	The target is the controlling process itself; that is, the client controls its own probes.														
direct mode	The target is a separate process; a client can either exec(2) a program or attach to a running process for probe control. The libtnfctl library uses proc(4) on the target process for probe and process control in this mode, and additionally provides basic process control features.														
indirect mode	The target is a separate process, but the controlling process is already using proc(4) to control the target, and hence libtnfctl cannot use those interfaces directly. Use this mode to control probes from within a debugger. In this mode, the client must provide a set of functions that libtnfctl can use to query and update the target process.														
kernel mode	The target is the Solaris kernel.														
tnfctl_internal_open()	INT														
tnfctl_exec_open()	D														
tnfctl_pid_open()	D														

tnfctl_indirect_open()			IND	
tnfctl_kernel_open()				K

These interfaces are used with the specified modes:

tnfctl_continue()		D		
tnfctl_probe_connect()	INT	D	IND	
tnfctl_probe_disconnect_all()	INT	D	IND	
tnfctl_trace_attrs_get()	INT	D	IND	K
tnfctl_buffer_alloc()	INT	D	IND	K
tnfctl_register_funcs()	INT	D	IND	K
tnfctl_probe_apply()	INT	D	IND	K
tnfctl_probe_apply_ids()	INT	D	IND	K
tnfctl_probe_state_get()	INT	D	IND	K
tnfctl_probe_enable()	INT	D	IND	K
tnfctl_probe_disable()	INT	D	IND	K
tnfctl_probe_trace()	INT	D	IND	K
tnfctl_probe_untrace()	INT	D	IND	K
tnfctl_check_libs()	INT	D	IND	K
tnfctl_close()	INT	D	IND	K
tnfctl_strerror()	INT	D	IND	K
tnfctl_buffer_dealloc()				K
tnfctl_trace_state_set()				K
tnfctl_filter_state_set()				K
tnfctl_filter_list_get()				K
tnfctl_filter_list_add()				K
tnfctl_filter_list_delete()				K

When using libtnfctl, the first task is to create a handle for controlling probes. The `tnfctl_internal_open()` function creates an internal mode handle for controlling probes in the same process, as described above. The `tnfctl_pid_open()` and `tnfctl_exec_open()` functions create handles in direct mode. The `tnfctl_indirect_open()` function creates an indirect mode handle, and the

libtnfctl(3TNF)

`tnfctl_kernel_open()` function creates a kernel mode handle. A handle is required for use in nearly all other `libtnfctl` functions. The `tnfctl_close()` function releases the resources associated with a handle.

The `tnfctl_continue()` function is used in direct mode to resume execution of the target process.

The `tnfctl_buffer_alloc()` function allocates a trace file or, in kernel mode, a trace buffer.

The `tnfctl_probe_apply()` and `tnfctl_probe_apply_ids()` functions call a specified function for each probe or for a designated set of probes.

The `tnfctl_register_funcs()` function registers functions to be called whenever new probes are seen or probes have disappeared, providing an opportunity to do one-time processing for each probe.

The `tnfctl_check_libs()` function is used primarily in indirect mode to check whether any new probes have appeared, that is, they have been made available by `dlopen(3DL)`, or have disappeared, that is, they have disassociated from the process by `dlclose(3DL)`.

The `tnfctl_probe_enable()` and `tnfctl_probe_disable()` functions control whether the probe, when hit, will be ignored.

The `tnfctl_probe_trace()` and `tnfctl_probe_untrace()` functions control whether an enabled probe, when hit, will cause an entry to be made in the trace file.

The `tnfctl_probe_connect()` and `tnfctl_probe_disconnect_all()` functions control which functions, if any, are called when an enabled probe is hit.

The `tnfctl_probe_state_get()` function returns information about the status of a probe, such as whether it is currently enabled.

The `tnfctl_trace_attrs_get()` function returns information about the tracing session, such as the size of the trace buffer or trace file.

The `tnfctl_strerror()` function maps a `tnfctl` error code to a string, for reporting purposes.

The remaining functions apply only to kernel mode.

The `tnfctl_trace_state_set()` function controls the master switch for kernel tracing. See `prex(1)` for more details.

The `tnfctl_filter_state_set()`, `tnfctl_filter_list_get()`, `tnfctl_filter_list_add()`, and `tnfctl_filter_list_delete()` functions allow a set of processes to be specified for which probes will not be ignored when hit. This prevents kernel activity caused by uninteresting processes from cluttering up the kernel's trace buffer.

	The <code>tnfctl_buffer_dealloc()</code> function deallocates the kernel's internal trace buffer.																																												
RETURN VALUES	Upon successful completion, these functions return <code>TNFCTL_ERR_NONE</code> .																																												
ERRORS	The error codes for <code>libtnfctl</code> are: <table> <tr> <td><code>TNFCTL_ERR_ACCES</code></td><td>Permission denied.</td></tr> <tr> <td><code>TNFCTL_ERR_NOTARGET</code></td><td>The target process completed.</td></tr> <tr> <td><code>TNFCTL_ERR_ALLOCFAIL</code></td><td>A memory allocation failure occurred.</td></tr> <tr> <td><code>TNFCTL_ERR_INTERNAL</code></td><td>An internal error occurred.</td></tr> <tr> <td><code>TNFCTL_ERR_SIZETOOSMALL</code></td><td>The requested trace size is too small.</td></tr> <tr> <td><code>TNFCTL_ERR_SIZETOOBIG</code></td><td>The requested trace size is too big.</td></tr> <tr> <td><code>TNFCTL_ERR_BADARG</code></td><td>Bad input argument.</td></tr> <tr> <td><code>TNFCTL_ERR_NOTDYNAMIC</code></td><td>The target is not a dynamic executable.</td></tr> <tr> <td><code>TNFCTL_ERR_NOLIBTNFPROBE</code></td><td><code>libtnfprobe.so</code> not linked in target.</td></tr> <tr> <td><code>TNFCTL_ERR_BUFBROKEN</code></td><td>Tracing is broken in the target.</td></tr> <tr> <td><code>TNFCTL_ERR_BUFEXISTS</code></td><td>A buffer already exists.</td></tr> <tr> <td><code>TNFCTL_ERR_NOBUF</code></td><td>No buffer exists.</td></tr> <tr> <td><code>TNFCTL_ERR_BADDEALLOC</code></td><td>Cannot deallocate buffer.</td></tr> <tr> <td><code>TNFCTL_ERR_NOPROCESS</code></td><td>No such target process exists.</td></tr> <tr> <td><code>TNFCTL_ERR_FILENOTFOUND</code></td><td>File not found.</td></tr> <tr> <td><code>TNFCTL_ERR_BUSY</code></td><td>Cannot attach to process or kernel because it is already tracing.</td></tr> <tr> <td><code>TNFCTL_ERR_INVALIDPROBE</code></td><td>Probe no longer valid.</td></tr> <tr> <td><code>TNFCTL_ERR_USR1</code></td><td>Error code reserved for user.</td></tr> <tr> <td><code>TNFCTL_ERR_USR2</code></td><td>Error code reserved for user.</td></tr> <tr> <td><code>TNFCTL_ERR_USR3</code></td><td>Error code reserved for user.</td></tr> <tr> <td><code>TNFCTL_ERR_USR4</code></td><td>Error code reserved for user.</td></tr> <tr> <td><code>TNFCTL_ERR_USR5</code></td><td>Error code reserved for user.</td></tr> </table>	<code>TNFCTL_ERR_ACCES</code>	Permission denied.	<code>TNFCTL_ERR_NOTARGET</code>	The target process completed.	<code>TNFCTL_ERR_ALLOCFAIL</code>	A memory allocation failure occurred.	<code>TNFCTL_ERR_INTERNAL</code>	An internal error occurred.	<code>TNFCTL_ERR_SIZETOOSMALL</code>	The requested trace size is too small.	<code>TNFCTL_ERR_SIZETOOBIG</code>	The requested trace size is too big.	<code>TNFCTL_ERR_BADARG</code>	Bad input argument.	<code>TNFCTL_ERR_NOTDYNAMIC</code>	The target is not a dynamic executable.	<code>TNFCTL_ERR_NOLIBTNFPROBE</code>	<code>libtnfprobe.so</code> not linked in target.	<code>TNFCTL_ERR_BUFBROKEN</code>	Tracing is broken in the target.	<code>TNFCTL_ERR_BUFEXISTS</code>	A buffer already exists.	<code>TNFCTL_ERR_NOBUF</code>	No buffer exists.	<code>TNFCTL_ERR_BADDEALLOC</code>	Cannot deallocate buffer.	<code>TNFCTL_ERR_NOPROCESS</code>	No such target process exists.	<code>TNFCTL_ERR_FILENOTFOUND</code>	File not found.	<code>TNFCTL_ERR_BUSY</code>	Cannot attach to process or kernel because it is already tracing.	<code>TNFCTL_ERR_INVALIDPROBE</code>	Probe no longer valid.	<code>TNFCTL_ERR_USR1</code>	Error code reserved for user.	<code>TNFCTL_ERR_USR2</code>	Error code reserved for user.	<code>TNFCTL_ERR_USR3</code>	Error code reserved for user.	<code>TNFCTL_ERR_USR4</code>	Error code reserved for user.	<code>TNFCTL_ERR_USR5</code>	Error code reserved for user.
<code>TNFCTL_ERR_ACCES</code>	Permission denied.																																												
<code>TNFCTL_ERR_NOTARGET</code>	The target process completed.																																												
<code>TNFCTL_ERR_ALLOCFAIL</code>	A memory allocation failure occurred.																																												
<code>TNFCTL_ERR_INTERNAL</code>	An internal error occurred.																																												
<code>TNFCTL_ERR_SIZETOOSMALL</code>	The requested trace size is too small.																																												
<code>TNFCTL_ERR_SIZETOOBIG</code>	The requested trace size is too big.																																												
<code>TNFCTL_ERR_BADARG</code>	Bad input argument.																																												
<code>TNFCTL_ERR_NOTDYNAMIC</code>	The target is not a dynamic executable.																																												
<code>TNFCTL_ERR_NOLIBTNFPROBE</code>	<code>libtnfprobe.so</code> not linked in target.																																												
<code>TNFCTL_ERR_BUFBROKEN</code>	Tracing is broken in the target.																																												
<code>TNFCTL_ERR_BUFEXISTS</code>	A buffer already exists.																																												
<code>TNFCTL_ERR_NOBUF</code>	No buffer exists.																																												
<code>TNFCTL_ERR_BADDEALLOC</code>	Cannot deallocate buffer.																																												
<code>TNFCTL_ERR_NOPROCESS</code>	No such target process exists.																																												
<code>TNFCTL_ERR_FILENOTFOUND</code>	File not found.																																												
<code>TNFCTL_ERR_BUSY</code>	Cannot attach to process or kernel because it is already tracing.																																												
<code>TNFCTL_ERR_INVALIDPROBE</code>	Probe no longer valid.																																												
<code>TNFCTL_ERR_USR1</code>	Error code reserved for user.																																												
<code>TNFCTL_ERR_USR2</code>	Error code reserved for user.																																												
<code>TNFCTL_ERR_USR3</code>	Error code reserved for user.																																												
<code>TNFCTL_ERR_USR4</code>	Error code reserved for user.																																												
<code>TNFCTL_ERR_USR5</code>	Error code reserved for user.																																												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:																																												

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc

libtnfctl(3TNF)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

SEE ALSO

prex(1), exec(2), dlclose(3DL), dlopen(3DL), TNF_PROBE(3TNF),
tnfctl_buffer_alloc(3TNF), tnfctl_buffer_dealloc(3TNF),
tnfctl_check_libs(3TNF), tnfctl_close(3TNF), tnfctl_continue(3TNF),
tnfctl_internal_open(3TNF), tnfctl_exec_open(3TNF),
tnfctl_filter_list_add(3TNF), tnfctl_filter_list_delete(3TNF),
tnfctl_filter_list_get(3TNF), tnfctl_filter_state_set(3TNF),
tnfctl_kernel_open(3TNF), tnfctl_pid_open(3TNF),
tnfctl_probe_apply(3TNF), tnfctl_probe_apply_ids(3TNF),
tnfctl_probe_connect(3TNF), tnfctl_probe_disable(3TNF),
tnfctl_probe_enable(3TNF), tnfctl_probe_state_get(3TNF),
tnfctl_probe_trace(3TNF), tnfctl_probe_untrace(3TNF),
tnfctl_indirect_open(3TNF), tnfctl_register_funcs(3TNF),
tnfctl_strerror(3TNF), tnfctl_trace_attrs_get(3TNF),
tnfctl_trace_state_set(3TNF), libtnfctl(3LIB), proc(4), attributes(5)

Linker and Libraries Guide

NOTES

This API is MT-Safe. Multiple threads may concurrently operate on independent tnfctl handles, which is the typical behavior expected. The libtnfctl library does not support multiple threads operating on the same tnfctl handle. If this is desired, it is the client's responsibility to implement locking to ensure that two threads that use the same tnfctl handle are not simultaneously in a libtnfctl interface.

NAME	log10 – base 10 logarithm function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double log10(double <i>x</i>);</pre>				
DESCRIPTION	The <code>log10()</code> function computes the base 10 logarithm of x , $\log_{10}(x)$. The value of x must be positive.				
RETURN VALUES	Upon successful completion, <code>log10()</code> returns the base 10 logarithm of x. If x is NaN, NaN is returned. If x is less than 0, <code>-HUGE_VAL</code> or NaN is returned, and <code>errno</code> is set to <code>EDOM</code>. If x is 0, <code>-HUGE_VAL</code> is returned and <code>errno</code> may be set to <code>ERANGE</code>. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>log10()</code> function will fail if: <code>EDOM</code> The value of x is negative. The <code>log10()</code> function may fail if: <code>ERANGE</code> The value of x is 0. No other errors will occur.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>log10()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>log(3M)</code> , <code>matherr(3M)</code> , <code>pow(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

log1p(3M)

NAME	log1p – compute natural logarithm				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double log1p(double <i>x</i>);</pre>				
DESCRIPTION	The <code>log1p()</code> function computes $\log_e(1.0 + x)$. The value of x must be greater than -1.0 .				
RETURN VALUES	Upon successful completion, <code>log1p()</code> returns the natural logarithm of $1.0 + x$. If x is NaN, <code>log1p()</code> returns NaN. If x is less than -1.0, <code>log1p()</code> returns <code>-HUGE_VAL</code> or NaN and sets <code>errno</code> to <code>EDOM</code>. If x is -1.0, <code>log1p()</code> returns <code>-HUGE_VAL</code> and may set <code>errno</code> to <code>ERANGE</code>. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>log1p()</code> function will fail if: <table><tr><td>EDOM</td><td>The value of x is less than -1.0.</td></tr></table> The <code>log1p()</code> function may fail and set <code>errno</code> to: <table><tr><td>ERANGE</td><td>The value of x is -1.0.</td></tr></table>	EDOM	The value of x is less than -1.0 .	ERANGE	The value of x is -1.0 .
EDOM	The value of x is less than -1.0 .				
ERANGE	The value of x is -1.0 .				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>log(3M)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

NAME	log – natural logarithm function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double log(double <i>x</i>);</pre>				
DESCRIPTION	The <code>log()</code> function computes the natural logarithm of x , $\log_e(x)$. The value of x must be positive.				
RETURN VALUES	Upon successful completion, <code>log()</code> returns the natural logarithm of x. If x is NaN, NaN is returned. If x is less than 0, <code>-HUGE_VAL</code> or NaN is returned and <code>errno</code> is set to <code>EDOM</code>. If x is 0, <code>-HUGE_VAL</code> is returned and <code>errno</code> may be set to <code>ERANGE</code>. In IEEE 754 mode (the <code>-Xlibmieee</code> <code>cc</code> compilation option), if x is Inf or a quiet NaN, x is returned; if x is a signaling NaN, a quiet NaN is returned and the invalid operation exception is raised; if x is 1, 0 is returned; for all other positive x, a normalized number is returned and the inexact exception is raised. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>log()</code> function will fail if: <code>EDOM</code> The value of x is negative. The <code>log()</code> function may fail if: <code>ERANGE</code> The value of x is 0. No other errors will occur.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>log()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>exp(3M)</code> , <code>isnan(3M)</code> , <code>log10(3M)</code> , <code>log1p(3M)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

logb(3M)

NAME	logb – radix-independent exponent				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double logb(double <i>x</i>) ;</pre>				
DESCRIPTION	The <code>logb()</code> function computes the exponent of x , which is the integral part of $\log_r x $, as a signed floating point value, for non-zero x , where r is the radix of the machine's floating-point arithmetic.				
RETURN VALUES	Upon successful completion, <code>logb()</code> returns the exponent of x. If x is 0.0, <code>logb()</code> returns <code>-HUGE_VAL</code> and sets <code>errno</code> to <code>EDOM</code>. If x is $\pm\text{Inf}$, <code>logb()</code> returns $+\text{Inf}$. If x is NaN, <code>logb()</code> returns NaN. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by various Standards.				
ERRORS	The <code>logb()</code> function will fail if: <table><tr><td>EDOM</td><td>The x argument is 0.0.</td></tr></table>	EDOM	The x argument is 0.0.		
EDOM	The x argument is 0.0.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>ilogb(3M)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code>				

NAME	maillock, mailunlock, touchlock – functions to manage lockfile(s) for user’s mailbox				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lmail [<i>library</i> ...] #include <maillock.h> int maillock(const char *<i>user</i>, int <i>retrycnt</i>) ; void mailunlock(void) ; void touchlock(void) ;</pre>				
DESCRIPTION	The maillock() function attempts to create a lockfile for the user’s mailfile. If a lockfile already exists, and it has not been modified in the last 5 minutes, maillock() will remove the lockfile and set its own lockfile. It is crucial that programs locking mail files refresh their locks at least every three minutes to maintain the lock. Refresh the lockfile by calling the touchlock() function with no arguments. The algorithm used to determine the age of the lockfile takes into account clock drift between machines using a network file system. A zero is written into the lockfile so that the lock will be respected by systems running the standard version of System V. If the lockfile has been modified in the last 5 minutes the process will sleep until the lock is available. The sleep algorithm is to sleep for 5 seconds times the attempt number. That is, the first sleep will be for 5 seconds, the next sleep will be for 10 seconds, etc. until the number of attempts reaches <i>retrycnt</i>. When the lockfile is no longer needed, it should be removed by calling mailunlock(). The <i>user</i> argument is the login name of the user for whose mailbox the lockfile will be created. maillock() assumes that user’s mailfiles are in the “standard” place as defined in <maillock.h>.				
RETURN VALUES	Upon successful completion, maillock() returns 0. Otherwise it returns -1.				
FILES	<table> <tr> <td>/var/mail/*</td><td>user mailbox files</td></tr> <tr> <td>/var/mail/*.lock</td><td>user mailbox lockfiles</td></tr> </table>	/var/mail/*	user mailbox files	/var/mail/*.lock	user mailbox lockfiles
/var/mail/*	user mailbox files				
/var/mail/*.lock	user mailbox lockfiles				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	libmail(3LIB), attributes(5)				

maillock(3MAIL)

NOTES	The mailunlock() function will only remove the lockfile created from the most previous call to maillock(). Calling maillock() for different users without intervening calls to mailunlock() will cause the initially created lockfile(s) to remain, potentially blocking subsequent message delivery until the current process finally terminates.

NAME	matherr – math library exception-handling function														
SYNOPSIS	<pre>#include <math.h> int matherr(struct exception *exc);</pre>														
DESCRIPTION	The The System V Interface Definition, Third Edition (SVID3) specifies that certain libm functions call <code>matherr()</code> when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named <code>matherr()</code> in their programs. The <code>matherr()</code> function is of the form described above. When an exception occurs, a pointer to the exception structure <code>exc</code> will be passed to the user-supplied <code>matherr()</code> function. This structure, which is defined in the <code><math.h></code> header file, is as follows: <pre>struct exception { int type; char *name; double arg1, arg2, retval; };</pre> The <code>type</code> member is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file): <table> <tr> <td>DOMAIN</td><td>argument domain exception</td></tr> <tr> <td>SING</td><td>argument singularity</td></tr> <tr> <td>OVERFLOW</td><td>overflow range exception</td></tr> <tr> <td>UNDERFLOW</td><td>underflow range exception</td></tr> <tr> <td>TLOSS</td><td>total loss of significance</td></tr> <tr> <td>PLOSS</td><td>partial loss of significance</td></tr> </table> Both <code>TLOSS</code> and <code>PLOSS</code> reflect limitations of particular algorithms for trigonometric functions that suffer abrupt declines in accuracy at definite boundaries. Since the implementation does not suffer such abrupt declines, <code>PLOSS</code> is never signaled. <code>TLOSS</code> is signaled for Bessel functions <i>only</i> to satisfy SVID3 requirements. The <code>name</code> member points to a string containing the name of the function that incurred the exception. The <code>arg1</code> and <code>arg2</code> members are the arguments with which the function was invoked. <code>retval</code> is set to the default value that will be returned by the function unless the user's <code>matherr()</code> sets it to a different value. If the user's <code>matherr()</code> function returns non-zero, no exception message will be printed and <code>errno</code> is not set. When an application is built as a SVID3 conforming application (see <code>standards(5)</code>), if <code>matherr()</code> is not supplied by the user, the default <code>matherr</code> exception-handling mechanisms, summarized in the table below, are invoked upon exception: <table> <tr> <td>DOMAIN</td><td>0.0 is usually returned, <code>errno</code> is set to <code>EDOM</code> and a message is usually printed on standard error.</td></tr> </table>	DOMAIN	argument domain exception	SING	argument singularity	OVERFLOW	overflow range exception	UNDERFLOW	underflow range exception	TLOSS	total loss of significance	PLOSS	partial loss of significance	DOMAIN	0.0 is usually returned, <code>errno</code> is set to <code>EDOM</code> and a message is usually printed on standard error.
DOMAIN	argument domain exception														
SING	argument singularity														
OVERFLOW	overflow range exception														
UNDERFLOW	underflow range exception														
TLOSS	total loss of significance														
PLOSS	partial loss of significance														
DOMAIN	0.0 is usually returned, <code>errno</code> is set to <code>EDOM</code> and a message is usually printed on standard error.														
SVID3 STANDARD CONFORMANCE															

matherr(3M)

**SVID3 ERROR
HANDLING
PROCEDURES
(compile with cc
-Xt)**

SING	The largest finite single-precision number, HUGE of appropriate sign, is returned, <code>errno</code> is set to <code>EDOM</code> , and a message is printed on standard error.
OVERFLOW	The largest finite single-precision number, HUGE of appropriate sign, is usually returned and <code>errno</code> is set to <code>ERANGE</code> .
UNDERFLOW	0.0 is returned and <code>errno</code> is set to <code>ERANGE</code> .
TLOSS	0.0 is returned, <code>errno</code> is set to <code>ERANGE</code> , and a message is printed on standard error.

In general, `errno` is not a reliable error indicator because it can be unexpectedly set by a function in a handler for an asynchronous signal.

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
<code>errno</code>	EDOM	EDOM	ERANGE	ERANGE	ERANGE
IEEE Exception	Invalid Operation	Division by Zero	Overflow	Underflow	–
<code>fp_exception_type</code>	<code>fp_invalid</code>	<code>fp_division</code>	<code>fp_overflow</code>	<code>fp_underflow</code>	–
ACOS, ASIN ($ x > 1$):	Md, 0.0	–	–	–	–
ACOSH ($x < 1$), ATANH ($ x > 1$):	NaN	–	–	–	–
ATAN2 (0,0):	Md, 0.0	–	–	–	–
COSH, SINH:	–	–	±HUGE	–	–
EXP:	–	–	+HUGE	0.0	–
FMOD ($x,0$):	x	–	–	–	–
HYPOT:	–	–	+HUGE	–	–
J0, J1, JN ($ x > X_TLOSS$):	–	–	–	–	Mt, 0.0
LGAMMA:					
usual cases	–	–	+HUGE	–	–
($x = 0$ or $-integer$)	–	Ms, +HUGE	–	–	–
LOG, LOG10:					
($x < 0$)	Md, –HUGE	–	–	–	–
($x = 0$)	–	Ms, –HUGE	–	–	–
POW:					

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
usual cases	–	–	±HUGE	±0.0	–
(x < 0) ** (y not an integer)	Md, 0.0	–	–	–	–
0 ** 0	Md, 0.0	–	–	–	–
0 ** (y < 0)	Md, 0.0	–	–	–	–
REMAINDER (x,0):	NaN	–	–	–	–
SCALB:	–	–	±HUGE_VAL	±0.0	–
SQRT (x < 0):	Md, 0.0	–	–	–	–
Y0, Y1, YN:					
(x < 0)	Md, –HUGE	–	–	–	–
(x = 0)	–	Md, –HUGE	–	–	–
(x > X_TLOSS)	–	–	–	–	Mt, 0.0

Abbreviations

Md	Message is printed (DOMAIN error).
Ms	Message is printed (SING error).
Mt	Message is printed (TLOSS error).
NaN	IEEE NaN result and invalid operation exception.
HUGE	Maximum finite single-precision floating-point number.
HUGE_VAL	IEEE ∞ result and division-by-zero exception.
X_TLOSS	The value X_TLOSS is defined in <values.h>.

The interaction of IEEE arithmetic and `matherr()` is not defined when executing under IEEE rounding modes other than the default round to nearest: `matherr()` is not always called on overflow or underflow and can return results that differ from those in this table.

**X/OPEN
COMMON
APPLICATION
ENVIRONMENT
(CAE)
SPECIFICATIONS
CONFORMANCE**

The X/Open System Interfaces and Headers (XSH) Issue 3 and later revisions of that specification no longer sanctions the use of the `matherr` interface. The following table summarizes the values returned in the exceptional cases. In general, XSH dictates that as long as one of the input argument(s) is a NaN, NaN is returned. In particular, `pow(NaN, 0) = NaN`.

**CAE
SPECIFICATION
ERROR
HANDLING
PROCEDURES
(compile with cc
-Xa)**

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
<code>errno</code>	EDOM	EDOM	ERANGE	ERANGE	ERANGE

matherr(3M)

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
ACOS, ASIN ($ x > 1$):	0.0	—	—	—	—
ATAN2 (0,0):	0.0	—	—	—	—
COSH, SINH:	—	—	{ $\pm$ HUGE_VAL}	—	—
EXP:	—	—	{+HUGE_VAL}	{0.0}	—
FMOD (x,0):	{NaN}	—	—	—	—
HYPOT:	—	—	{+HUGE_VAL}	—	—
J0, J1, JN ($ x > X_TLOSS$):	—	—	—	—	{0.0}
LGAMMA: usual cases	—	—	{+HUGE_VAL}	—	—
(x = 0 or —integer)	—	+HUGE_VAL	—	—	—
LOG, LOG10: (x < 0)	—HUGE_VAL	—	—	—	—
(x = 0)	—	—HUGE_VAL	—	—	—
POW: usual cases	—	—	$\pm$ HUGE_VAL	± 0.0	—
(x < 0) ** (y not an integer)	0.0	—	—	—	—
0 ** 0	{1.0}	—	—	—	—
0 ** (y < 0)	{—HUGE_VAL}	—	—	—	—
SQRT (x < 0):	0.0	—	—	—	—
Y0, Y1, YN: (x < 0)	{—HUGE_VAL}	—	—	—	—
(x = 0)	—	{—HUGE_VAL}	—	—	—

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
(x > X_TLOSS)	—	—	—	—	0.0

Abbreviations	{...}	errno is not to be relied upon in all braced cases.
	NaN	IEEE NaN result and invalid operation exception.
	HUGE_VAL	IEEE ∞ result and division-by-zero exception.
	X_TLOSS	The value X_TLOSS is defined in <values.h>.

ANSI/ISO-C STANDARD CONFORMANCE The ANSI/ISO-C standard covers a small subset of the CAE specification.

The following table summarizes the values returned in the exceptional cases.

ANSI/ISO-C ERROR HANDLING PROCEDURES
(compile with cc -xc)

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW
errno	EDOM	EDOM	ERANGE	ERANGE
ACOS, ASIN ($ x > 1$):	0.0	—	—	—
ATAN2 (0,0):	0.0	—	—	—
EXP:	—	—	+HUGE_VAL	0.0
FMOD (x,0):	NaN	—	—	—
LOG, LOG10:				
(x < 0)	-HUGE_VAL	—	—	—
(x = 0)	—	-HUGE_VAL	—	—
POW:				
usual cases	—	—	$\pm$ HUGE_VAL	± 0.0
(x < 0) ** (y not an integer)	0.0	—	—	—
0 ** (y < 0)	-HUGE_VAL	—	—	—
SQRT (x < 0):	0.0	—	—	—

ABBREVIATIONS	NaN	IEEE NaN result and invalid operation exception.
	HUGE_VAL	IEEE ∞ result and division-by-zero.

EXAMPLES **EXAMPLE 1** Example of matherr() function

```
#include <stdio.h>
#include <stdlib.h>
```

matherr(3M)

EXAMPLE 1 Example of matherr() function (Continued)

```
#include <math.h>

int
matherr(struct exception *x) {
    switch (x->type) {
        case DOMAIN:
            /* change sqrt to return sqrt(-arg1), not NaN */
            if (!strcmp(x->name, "sqrt")) {
                x->retval = sqrt(-x->arg1);
                return (0); /* print message and set errno */
            } /* FALLTHRU */
        case SING:
            /* all other domain or sing exceptions, print message and */
            /* abort */
            fprintf(stderr, "domain exception in %s\n", x->name);
            abort();
            break;
    }
    return (0); /* all other exceptions, execute default procedure */
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO attributes(5), standards(5)

NAME	m_create_layout – initialize a layout object								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llayout [<i>library...</i>] #include <sys/layout.h></pre> LayoutObject m_create_layout(const AttrObject <i>attrobj</i>, const char* <i>modifier</i>);								
DESCRIPTION	The <code>m_create_layout()</code> function creates a LayoutObject associated with the locale identified by <i>attrobj</i>. The LayoutObject is an opaque object containing all the data and methods necessary to perform the layout operations on context-dependent or directional characters of the locale identified by the <i>attrobj</i>. The memory for the LayoutObject is allocated by <code>m_create_layout()</code>. The LayoutObject created has default layout values. If the <i>modifier</i> argument is not NULL, the layout values specified by the <i>modifier</i> overwrite the default layout values associated with the locale. Internal states maintained by the layout transformation function across transformations are set to their initial values. The <i>attrobj</i> argument is or may be an amalgam of many opaque objects. A locale object is just one example of the type of object that can be attached to an attribute object. The <i>attrobj</i> argument specifies a name that is usually associated with a locale category. If <i>attrobj</i> is NULL, the created LayoutObject is associated with the current locale as set by the <code>setlocale(3C)</code> function. The <i>modifier</i> argument announces a set of layout values when the LayoutObject is created.								
RETURN VALUES	Upon successful completion, the <code>m_create_layout()</code> function returns a LayoutObject for use in subsequent calls to <code>m_*_layout()</code> functions. Otherwise the <code>m_create_layout()</code> function returns (LayoutObject) 0 and sets <code>errno</code> to indicate the error.								
ERRORS	The <code>m_create_layout()</code> function may fail if: <table> <tr> <td>EBADF</td><td>The attribute object is invalid or the locale associated with the attribute object is not available.</td></tr> <tr> <td>EINVAL</td><td>The <i>modifier</i> string has a syntax error or it contains unknown layout values.</td></tr> <tr> <td>EMFILE</td><td>There are {OPEN_MAX} file descriptors currently open in the calling process.</td></tr> <tr> <td>ENOMEM</td><td>Insufficient storage space is available.</td></tr> </table>	EBADF	The attribute object is invalid or the locale associated with the attribute object is not available.	EINVAL	The <i>modifier</i> string has a syntax error or it contains unknown layout values.	EMFILE	There are {OPEN_MAX} file descriptors currently open in the calling process.	ENOMEM	Insufficient storage space is available.
EBADF	The attribute object is invalid or the locale associated with the attribute object is not available.								
EINVAL	The <i>modifier</i> string has a syntax error or it contains unknown layout values.								
EMFILE	There are {OPEN_MAX} file descriptors currently open in the calling process.								
ENOMEM	Insufficient storage space is available.								

m_create_layout(3LAYOUT)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO setlocale(3C), attributes(5)

NAME	md5, md5_calc, MD5Init, MD5Update, MD5Final – MD5 digest functions
SYNOPSIS	<pre>cc [flag ...] file ... -lmd5 [library ...] #include <md5.h> void md5_calc(unsigned char *output, unsigned char *input, unsigned int inlen); void MD5Init(MD5_CTX *context); void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inlen); void MD5Final(unsigned char *output, MD5_CTX *context);</pre>
DESCRIPTION	These functions implement the MD5 message-digest algorithm, which takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is intended for digital signature applications, where large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA. md5_calc() The <code>md5_calc()</code> function computes an MD5 digest on a single message block. The <i>inlen</i>-byte block is pointed to by <i>input</i>, and the 16-byte MD5 digest is written to <i>output</i>. MD5Init(), MD5Update(), MD5Final() The <code>MD5Init()</code>, <code>MD5Update()</code>, and <code>MD5Final()</code> functions allow an MD5 digest to be computed over multiple message blocks; between blocks, the state of the MD5 computation is held in an MD5 context structure, allocated by the caller. A complete digest computation consists of one call to <code>MD5Init()</code>, one or more calls to <code>MD5Update()</code>, and one call to <code>MD5Final()</code>, in that order. The <code>MD5Init()</code> function initializes the MD5 context structure pointed to by <i>context</i>. The <code>MD5Update()</code> function computes a partial MD5 digest on the <i>inlen</i>-byte message block pointed to by <i>input</i>, and updates the MD5 context structure pointed to by <i>context</i> accordingly. The <code>MD5Final()</code> function generates the final MD5 digest, using the MD5 context structure pointed to by <i>context</i>; the 16-byte MD5 digest is written to <i>output</i>. After calling <code>MD5Final()</code>, the state of the context structure is undefined; it must be reinitialized with <code>MD5Init()</code> before being used again.
RETURN VALUES	These functions do not return a value.
EXAMPLES	EXAMPLE 1 Authenticate a message found in multiple buffers The following is a sample function that must authenticate a message that is found in multiple buffers. The calling function provides an authentication buffer that will contain the result of the MD5 digest. <pre>int AuthenticateMsg(unsigned char *auth_buffer, struct iovec *messageIov, unsigned int num_buffers) {</pre>

md5(3EXT)

EXAMPLE 1 Authenticate a message found in multiple buffers (Continued)

```
MD5_CTX md5_context;
unsigned int i;

MD5Init(&md5_context);

for(i=0, i<num_buffers; i++)
{
    MD5Update(&md5_context, messageIov->iiov_base,
              messageIov->iiov_len);
    messageIov += sizeof(struct iovec);
}

MD5Final(auth_buffer, &md5_context);

return 0;
}
```

EXAMPLE 2 Use `md5_calc()` to generate the MD5 digest

Since the buffer to be computed is contiguous, the `md5_calc()` function can be used to generate the MD5 digest.

```
int AuthenticateMsg(unsigned char *auth_buffer, unsigned
                    char *buffer, unsigned int length)
{
    md5_calc(buffer, auth_buffer, length);

    return (0);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe

SEE ALSO `libmd5(3LIB)`

Rivest, R., The MD5 Message-Digest Algorithm, RFC 1321, April 1992.

NAME	m_destroy_layout – destroy a layout object				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llayout [<i>library...</i>] #include <sys/layout.h> int m_destroy_layout(const LayoutObject <i>layoutobject</i>);</pre>				
DESCRIPTION	The m_destroy_layout() function destroys a LayoutObject by deallocating the layout object and all the associated resources previously allocated by the m_create_layout(3LAYOUT) function.				
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise -1 is returned and errno is set to indicate the error.				
ERRORS	The m_destroy_layout() function may fail if: <table> <tr> <td>EBADF</td><td>The attribute object is erroneous.</td></tr> <tr> <td>EFAULT</td><td>Errors occurred while processing the request.</td></tr> </table>	EBADF	The attribute object is erroneous.	EFAULT	Errors occurred while processing the request.
EBADF	The attribute object is erroneous.				
EFAULT	Errors occurred while processing the request.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	m_create_layout(3LAYOUT), attributes(5)				

media_findname(3VOLMGT)

NAME	media_findname – convert a supplied name into an absolute pathname that can be used to access removable media										
SYNOPSIS	<pre>cc [flag ...] file ... -lvolumgt [library ...] #include <volumgt.h> char *media_findname(char *start);</pre>										
DESCRIPTION	media_findname() converts the supplied <i>start</i> string into an absolute pathname that can then be used to access a particular piece of media. The <i>start</i> parameter can be one of the following types of specifications: <table> <tr> <td>/dev/...</td><td>An absolute pathname in /dev, such as /dev/rdiskette0, in which case a copy of that string is returned (see NOTES on this page).</td></tr> <tr> <td>/vol/...</td><td>An absolute Volume Management pathname, such as /vol/dev/aliases/floppy0 or /vol/dsk/fred. If this supplied pathname is not a symbolic link, then a copy of that pathname is returned. If the supplied pathname is a symbolic link then it is dereferenced and a copy of that dereferenced pathname is returned.</td></tr> <tr> <td>volume_name</td><td>The Volume Management volume name for a particular volume, such as fred (see fdformat(1) for a description of how to label floppies). In this case a pathname in the Volume Management namespace is returned.</td></tr> <tr> <td>volumgt_symname</td><td>The Volume Management symbolic name for a device, such as floppy0 or cdrom2 (see volfs(7FS) for more information on Volume Management symbolic names), in which case a pathname in the Volume Management namespace is returned.</td></tr> <tr> <td>media_type</td><td>The Volume Management generic media type name. For example, floppy or cdrom. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the pathname to the volume found is returned.</td></tr> </table>	/dev/...	An absolute pathname in /dev, such as /dev/rdiskette0, in which case a copy of that string is returned (see NOTES on this page).	/vol/...	An absolute Volume Management pathname, such as /vol/dev/aliases/floppy0 or /vol/dsk/fred. If this supplied pathname is not a symbolic link, then a copy of that pathname is returned. If the supplied pathname is a symbolic link then it is dereferenced and a copy of that dereferenced pathname is returned.	volume_name	The Volume Management volume name for a particular volume, such as fred (see fdformat(1) for a description of how to label floppies). In this case a pathname in the Volume Management namespace is returned.	volumgt_symname	The Volume Management symbolic name for a device, such as floppy0 or cdrom2 (see volfs(7FS) for more information on Volume Management symbolic names), in which case a pathname in the Volume Management namespace is returned.	media_type	The Volume Management generic media type name. For example, floppy or cdrom. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the pathname to the volume found is returned.
/dev/...	An absolute pathname in /dev, such as /dev/rdiskette0, in which case a copy of that string is returned (see NOTES on this page).										
/vol/...	An absolute Volume Management pathname, such as /vol/dev/aliases/floppy0 or /vol/dsk/fred. If this supplied pathname is not a symbolic link, then a copy of that pathname is returned. If the supplied pathname is a symbolic link then it is dereferenced and a copy of that dereferenced pathname is returned.										
volume_name	The Volume Management volume name for a particular volume, such as fred (see fdformat(1) for a description of how to label floppies). In this case a pathname in the Volume Management namespace is returned.										
volumgt_symname	The Volume Management symbolic name for a device, such as floppy0 or cdrom2 (see volfs(7FS) for more information on Volume Management symbolic names), in which case a pathname in the Volume Management namespace is returned.										
media_type	The Volume Management generic media type name. For example, floppy or cdrom. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the pathname to the volume found is returned.										
RETURN VALUES	Upon successful completion media_findname() returns a pointer to the pathname found. In the case of an error a null pointer is returned.										

ERRORS For cases where the supplied *start* parameter is an absolute pathname, `media_findname()` can fail, returning a null string pointer, if an `lstat(2)` of that supplied pathname fails. Also, if the supplied absolute pathname is a symbolic link, `media_findname()` can fail if a `readlink(2)` of that symbolic link fails, or if a `stat(2)` of the pathname pointed to by that symbolic link fails, or if any of the following is true:

ENXIO The specified absolute pathname was not a character special device, and it was not a directory with a character special device in it.

EXAMPLES **EXAMPLE 1** Sample programs of the `media_findname()` function.

The following example attempts to find what the Volume Management pathname is to a piece of media called fred. Notice that a `volmgt_check()` is done first (see the **NOTES** section on this page).

```
(void) volmgt_check(NULL);
if ((nm = media_findname("fred")) != NULL) {
    (void) printf("media named \"fred\" is at \"%s\"\n", nm);
} else {
    (void) printf("media named \"fred\" not found\n");
}
```

This example looks for whatever volume is in the first floppy drive, letting `media_findname()` call `volmgt_check()` if and only if no floppy is currently known to be the first floppy drive.

```
if ((nm = media_findname("floppy0")) != NULL) {
    (void) printf("path to floppy0 is \"%s\"\n", nm);
} else {
    (void) printf("nothing in floppy0\n");
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Unsafe

SEE ALSO `cc(1B)`, `fdformat(1)`, `vold(1M)`, `lstat(2)`, `readlink(2)`, `stat(2)`, `free(3C)`, `malloc(3C)`, `volmgt_check(3VOLMGT)`, `volmgt_inuse(3VOLMGT)`, `volmgt_root(3VOLMGT)`, `volmgt_running(3VOLMGT)`, `volmgt_symname(3VOLMGT)`, `attributes(5)`, `volfs(7FS)`

NOTES If `media_findname()` cannot find a match for the supplied name, it performs a `volmgt_check(3VOLMGT)` and tries again, so it can be more efficient to perform `volmgt_check()` before calling `media_findname()`.

`media_findname(3VOLMGT)`

Upon success `media_findname()` returns a pointer to string which has been allocated; this should be freed when no longer in use (see `free(3C)`).

NAME	media_getattr, media_setattr – get and set media attributes																																
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> char *media_getattr(char *vol_path, char *attr); int media_setattr(char *vol_path, char *attr, char *value);</pre>																																
DESCRIPTION	media_setattr() and media_getattr() respectively set and get attribute-value pairs (called properties) on a per-volume basis. Volume Management supports system properties and user properties. System properties are ones that Volume Management predefines. Some of these system properties are writable, but only by the user that owns the volume being specified, and some system properties are read only: <table><tr><th>Attribute</th><th>Writable</th><th>Value</th><th>Description</th></tr><tr><td>s-access</td><td>RO</td><td>"seq", "rand"</td><td>sequential or random access</td></tr><tr><td>s-density</td><td>RO</td><td>"low", "medium", "high"</td><td>media density</td></tr><tr><td>s-parts</td><td>RO</td><td>comma separated list of slice numbers</td><td>list of partitions on this volume</td></tr><tr><td>s-location</td><td>RO</td><td>pathname</td><td>Volume Management pathname to media</td></tr><tr><td>s-mejectable</td><td>RO</td><td>"true", "false"</td><td>whether or not media is manually ejectable</td></tr><tr><td>s-rmoneject</td><td>R/W</td><td>"true", "false"</td><td>should media access points be removed from database upon ejection</td></tr><tr><td>s-enxio</td><td>R/W</td><td>"true", "false"</td><td>if set return ENXIO when media access attempted</td></tr></table> Properties can also be defined by the user. In this case the value can be any string the user wishes.	Attribute	Writable	Value	Description	s-access	RO	"seq", "rand"	sequential or random access	s-density	RO	"low", "medium", "high"	media density	s-parts	RO	comma separated list of slice numbers	list of partitions on this volume	s-location	RO	pathname	Volume Management pathname to media	s-mejectable	RO	"true", "false"	whether or not media is manually ejectable	s-rmoneject	R/W	"true", "false"	should media access points be removed from database upon ejection	s-enxio	R/W	"true", "false"	if set return ENXIO when media access attempted
Attribute	Writable	Value	Description																														
s-access	RO	"seq", "rand"	sequential or random access																														
s-density	RO	"low", "medium", "high"	media density																														
s-parts	RO	comma separated list of slice numbers	list of partitions on this volume																														
s-location	RO	pathname	Volume Management pathname to media																														
s-mejectable	RO	"true", "false"	whether or not media is manually ejectable																														
s-rmoneject	R/W	"true", "false"	should media access points be removed from database upon ejection																														
s-enxio	R/W	"true", "false"	if set return ENXIO when media access attempted																														
RETURN VALUES	Upon successful completion media_getattr() returns a pointer to the value corresponding to the specified attribute. A null pointer is returned if the specified volume doesn't exist, if the specified attribute for that volume doesn't exist, if the specified attribute is boolean and its value is false, or if malloc(3C) fails to allocate space for the return value. media_setattr() returns 1 upon success, and 0 upon failure.																																

media_getattr(3VOLMGT)

ERRORS Both `media_getattr()` and `media_setattr()` can fail returning a null pointer if an `open(2)` of the specified *vol_path* fails, if an `fstat(2)` of that pathname fails, or if that pathname is not a block or character special device.

`media_getattr()` can also fail if the specified attribute was not found, and `media_setattr()` can also fail if the caller doesn't have permission to set the attribute, either because it's a system attribute, or because the caller doesn't own the specified volume.

Additionally, either routine can fail returning the following error values:

ENXIO	The Volume Management daemon, <i>vold</i> , is not running
EINTR	The routine was interrupted by the user before finishing

EXAMPLES **EXAMPLE 1** Using `media_getattr()`

The following example checks to see if the volume called *fred* that Volume Management is managing can be ejected by means of software, or if it can only be manually ejected:

```
if (media_getattr("/vol/rdisk/fred", "s-mejectable") != NULL) {
    (void) printf("\nfred\n must be manually ejected\n");
} else {
    (void) printf("software can eject \nfred\n\n");
}
```

This example shows setting the *s-enxio* property for the floppy volume currently in the first floppy drive:

```
int      res;
if ((res = media_setattr("/vol/dev/aliases/floppy0", "s-enxio",
    "true")) == 0) {
    (void) printf("can't set s-enxio flag for floppy0\n");
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `cc(1B)`, `vold(1M)`, `lstat(2)`, `open(2)`, `readlink(2)`, `stat(2)`, `free(3C)`, `malloc(3C)`, `media_findname(3VOLMGT)`, `volmgt_check(3VOLMGT)`, `volmgt_inuse(3VOLMGT)`, `volmgt_root(3VOLMGT)`, `volmgt_running(3VOLMGT)`, `volmgt_symname(3VOLMGT)`, `attributes(5)`

NOTES Upon success `media_getattr()` returns a pointer to a string which has been allocated, and should be freed when no longer in use (see `free(3C)`).

NAME	media_getid – return the id of a piece of media						
SYNOPSIS	<pre>cc [flag ...] file ...-lvolgmt [library ...] #include <volmgt.h> ulonglong_t media_getid(char *vol_path) ;</pre>						
DESCRIPTION	media_getid() returns the <i>id</i> of a piece of media. Volume Management must be running. See volmgt_running(3VOLMGT).						
PARAMETERS	<i>vol_path</i> Path to the block or character special device.						
RETURN VALUES	media_getid() returns the <i>id</i> of the volume. This value is unique for each volume. If media_getid() returns 0, the <i>path</i> provided is not valid, for example, it is a block or char device.						
EXAMPLES	EXAMPLE 1 Using media_getid() The following example first checks if Volume Management is running, then checks the volume management name space for <i>path</i>, and then returns the <i>id</i> for the piece of media. <pre>char *path; ... if (volmgt_running()) { if (volmgt_ownspath(path)) { (void) printf("id of %s is %lld\n", path, media_getid(path)); } }</pre> If a program using media_getid() does not check whether or not Volume Management is running, then any NULL return value will be ambiguous, as it could mean that either Volume Management does not have <i>path</i> in its name space, or Volume Management is not running.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT Level</td><td>Safe</td></tr> <tr> <td>Commitment Level</td><td>Public</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT Level	Safe	Commitment Level	Public
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT Level	Safe						
Commitment Level	Public						
SEE ALSO	volmgt_ownspath(3VOLMGT), volmgt_running(3VOLMGT), attributes(5)						

m_getvalues_layout(3LAYOUT)

NAME	m_getvalues_layout – query layout values of a LayoutObject				
SYNOPSIS	<pre>cc [flag...] file... -llayout [library...] #include <sys/layout.h> int m_getvalues_layout(const LayoutObject layout_object, LayoutValues values, int *index_returned);</pre>				
DESCRIPTION	The <code>m_getvalues_layout()</code> function queries the current setting of layout values within a <code>LayoutObject</code>. The <code>layout_object</code> argument specifies a <code>LayoutObject</code> returned by the <code>m_create_layout(3LAYOUT)</code> function. The <code>values</code> argument specifies the list of layout values that are to be queried. Each value element of a <code>LayoutValueRec</code> must point to a location where the layout value is stored. That is, if the layout value is of type <code>T</code>, the argument must be of type <code>T*</code>. The values are queried from the <code>LayoutObject</code> and represent its current state. It is the user's responsibility to manage the space allocation for the layout values queried. If the layout value name has <code>QueryValueSize</code> OR-ed to it, instead of the value of the layout value, only its size is returned. The caller can use this option to determine the amount of memory needed to be allocated for the layout values queried.				
RETURN VALUES	Upon successful completion, the <code>m_getvalues_layout()</code> function returns 0. If any value cannot be queried, the index of the value causing the error is returned in <code>index_returned</code> , -1 is returned and <code>errno</code> is set to indicate the error.				
ERRORS	The <code>m_getvalues_layout()</code> function may fail if: <table><tr><td>EINVAL</td><td>The layout value specified by <code>index_returned</code> is unknown, its value is invalid, or the <code>layout_object</code> argument is invalid. In the case of an invalid <code>layout_object</code> argument, the value returned in <code>index_returned</code> is -1.</td></tr></table>	EINVAL	The layout value specified by <code>index_returned</code> is unknown, its value is invalid, or the <code>layout_object</code> argument is invalid. In the case of an invalid <code>layout_object</code> argument, the value returned in <code>index_returned</code> is -1.		
EINVAL	The layout value specified by <code>index_returned</code> is unknown, its value is invalid, or the <code>layout_object</code> argument is invalid. In the case of an invalid <code>layout_object</code> argument, the value returned in <code>index_returned</code> is -1.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>m_create_layout(3LAYOUT)</code> , <code>attributes(5)</code>				

NAME	mkdirm, rmdirm – create or remove directories in a path				
SYNOPSIS	<pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> int mkdirm(const char *path, mode_t mode); int rmdirm(char *dir, char *dir1);</pre>				
DESCRIPTION	The mkdirm() function creates all the missing directories in <i>path</i> with <i>mode</i>. See chmod(2) for the values of <i>mode</i>. The rmdirm() function removes directories in path <i>dir</i>. This removal begins at the end of the path and moves backward toward the root as far as possible. If an error occurs, the remaining path is stored in <i>dir1</i>.				
RETURN VALUES	If <i>path</i> already exists or if a needed directory cannot be created, mkdirm() returns –1 and sets <i>errno</i> to one of the error values listed for mkdirm(2). It returns zero if all the directories are created. The rmdirm() function returns 0 if it is able to remove every directory in the path. It returns –2 if a “.” or “..” is in the path and –3 if an attempt is made to remove the current directory. Otherwise it returns –1.				
EXAMPLES	EXAMPLE 1 Example of creating scratch directories. The following example creates scratch directories. <pre>/* create scratch directories */ if(mkdirm("/tmp/sub1/sub2/sub3", 0755) == -1) { fprintf(stderr, "cannot create directory"); exit(1); } chdir("/tmp/sub1/sub2/sub3"); . . . /* cleanup */ chdir("/tmp"); rmdirm("sub1/sub2/sub3");</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	chmod(2), mkdirm(2), rmdirm(2), malloc(3C), attributes(5)				
NOTES	mkdirm() uses malloc(3C) to allocate temporary space for the string.				

mkdirp(3GEN)

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

NAME	mp, mp_madd, mp_msub, mp_mult, mp_mdiv, mp_mcmp, mp_min, mp_mout, mp_pow, mp_gcd, mp_rpow, mp_itom, mp_xtom, mp_mtox, mp_mfree – multiple precision integer arithmetic
SYNOPSIS	<pre>cc [flag ...] file ... -lmp [library ...] #include <mp.h> void mp_madd(MINT *a, MINT *b, MINT *c); void mp_msub(MINT *a, MINT *b, MINT *c); void mp_mult(MINT *a, MINT *b, MINT *c); void mp_mdiv(MINT *a, MINT *b, MINT *q, MINT *r); int mp_mcmp(MINT *a, MINT *b); int mp_min(MINT *a); void mp_mout(MINT *a); void mp_pow(MINT *a, MINT *b, MINT *c, MINT *d); void mp_gcd(MINT *a, MINT *b, MINT *c); void mp_rpow(MINT *a, short n, MINT *b); int mp_msqrt(MINT *a, MINT *b, MINT *r); void mp_sdiv(MINT *a, short n, MINT *q, short *r); MINT * mp_itom(short n); MINT * mp_xtom(char *a); char * mp_mtox(MINT *a); void mp_mfree(MINT *a);</pre>
DESCRIPTION	These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type MINT. Pointers to a MINT should be initialized using the function <code>mp_itom(n)</code>, which sets the initial value to <i>n</i>. Alternatively, <code>mp_xtom(a)</code> may be used to initialize a MINT from a string of hexadecimal digits. <code>mp_mfree(a)</code> may be used to release the storage allocated by the <code>mp_itom(a)</code> and <code>mp_xtom(a)</code> routines. The <code>mp_madd(a,b,c)</code>, <code>mp_msub(a,b,c)</code> and <code>mp_mult(a,b,c)</code> functions assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. The <code>mp_mdiv(a,b,q,r)</code> function assigns the quotient and remainder, respectively, to its third and fourth arguments. The <code>mp_sdiv(a,n,q,r)</code> function is similar to <code>mp_mdiv(a,b,q,r)</code> except that the divisor is an ordinary integer. The <code>mp_msqrt(a,b,r)</code> function produces the square root and remainder of its first argument. The <code>mp_mcmp(a,b)</code> function compares the values of its arguments and returns 0 if the two values are equal, a value greater than 0 if the first argument is greater than the second, and a value less than 0 if the second argument is greater than the first. The <code>mp_rpow(a,n,b)</code> function raises <i>a</i> to the <i>n</i>th power and assigns this value to <i>b</i>. The

mp(3MP)

`mp_pow(a,b,c,d)` function raises *a* to the *b*th power, reduces the result modulo *c* and assigns this value to *d*. The `mp_min(a)` and `mp_mout(a)` functions perform decimal input and output. The `mp_gcd(a,b,c)` function finds the greatest common divisor of the first two arguments, returning it in the third argument. The `mp_mtox(a)` function provides the inverse of `mp_xtom(a)`. To release the storage allocated by `mp_mtox(a)` use `free()` (see `malloc(3C)`).

Use the `-lmp` loader option to obtain access to these functions.

FILES `/usr/lib/libmp.a`
`/usr/lib/libmp.so`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `exp(3M)`, `malloc(3C)`, `libmp(3LIB)`, `attributes(5)`

DIAGNOSTICS Illegal operations and running out of memory produce messages and core images.

WARNINGS The function `pow()` exists in both `libmp` and `libm` with widely differing semantics. This is why `libmp.so.2` exists. `libmp.so.1` exists solely for reasons of backward compatibility, and should not be used otherwise. Use the `mp_*( )` functions instead. See `libmp(3LIB)`.

NAME	m_setvalues_layout – set layout values of a LayoutObject				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -llayout [<i>library...</i>] #include <sys/layout.h> int m_setvalues_layout(LayoutObject <i>layout_object</i>, const LayoutValues <i>values</i>, int *<i>index_returned</i>);</pre>				
DESCRIPTION	The <code>m_setvalues_layout()</code> function changes the layout values of a LayoutObject. The <i>layout_object</i> argument specifies a LayoutObject returned by the <code>m_create_layout(3LAYOUT)</code> function. The <i>values</i> argument specifies the list of layout values that are to be changed. The values are written into the LayoutObject and may affect the behavior of subsequent layout functions. Some layout values do alter internal states maintained by a LayoutObject. The <code>m_setvalues_layout()</code> function can be implemented as a macro that evaluates the first argument twice.				
RETURN VALUES	Upon successful completion, the requested layout values are set and 0 is returned. Otherwise -1 is returned and <code>errno</code> is set to indicate the error. If any value cannot be set, none of the layout values are changed and the (zero-based) index of the first value causing the error is returned in <i>index_returned</i>.				
ERRORS	The <code>m_setvalues_layout()</code> function may fail if: <table> <tr> <td>EINVAL</td><td>The layout value specified by <i>index_returned</i> is unknown, its value is invalid, or the <i>layout_object</i> argument is invalid.</td></tr> <tr> <td>EMFILE</td><td>There are {OPEN_MAX} file descriptors currently open in the calling process.</td></tr> </table>	EINVAL	The layout value specified by <i>index_returned</i> is unknown, its value is invalid, or the <i>layout_object</i> argument is invalid.	EMFILE	There are {OPEN_MAX} file descriptors currently open in the calling process.
EINVAL	The layout value specified by <i>index_returned</i> is unknown, its value is invalid, or the <i>layout_object</i> argument is invalid.				
EMFILE	There are {OPEN_MAX} file descriptors currently open in the calling process.				
USAGE	Do not use expressions with side effects such as auto-increment or auto-decrement within the first argument to the <code>m_setvalues_layout()</code> function.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>m_create_layout(3LAYOUT)</code> , <code>attributes(5)</code>				

m_transform_layout(3LAYOUT)

NAME	m_transform_layout – layout transformation								
SYNOPSIS	<pre>cc [flag...] file... -llayout [library...] #include <sys/layout.h> int m_transform_layout(LayoutObject <i>layout_object</i>, const char *<i>InpBuf</i>, const size_t <i>ImpSize</i>, const void *<i>OutBuf</i>, size_t *<i>Outsize</i>, size_t *<i>InpToOut</i>, size_t *<i>OutToInp</i>, unsigned char *<i>Property</i>, size_t *<i>InpBufIndex</i>) ;</pre>								
DESCRIPTION	The <code>m_transform_layout()</code> function performs layout transformations (reordering, shaping, cell determination) or provides additional information needed for layout transformation (such as the expected size of the transformed layout, the nesting level of different segments in the text and cross-references between the locations of the corresponding elements before and after the layout transformation). Both the input text and output text are character strings. The <code>m_transform_layout()</code> function transforms the input text in <i>InpBuf</i> according to the current layout values in <i>layout_object</i>. Any layout value whose value type is <code>LayoutTextDescriptor</code> describes the attributes of the <i>InpBuf</i> and <i>OutBuf</i> arguments. If the attributes are the same for both <i>InpBuf</i> and <i>OutBuf</i>, a null transformation is performed with respect to that specific layout value. The <i>InpBuf</i> argument specifies the source text to be processed. The <i>InpBuf</i> may not be NULL, unless there is a need to reset the internal state. The <i>ImpSize</i> argument is the number of bytes within <i>InpBuf</i> to be processed by the transformation. Its value will not change after return from the transformation. <i>ImpSize</i> set to -1 indicates that the text in <i>InpBuf</i> is delimited by a null code element. If <i>ImpSize</i> is not set to -1, it is possible to have some null elements in the input buffer. This might be used, for example, for a “one shot” transformation of several strings, separated by nulls. Output of this function may be one or more of the following depending on the setting of the arguments: <table> <tr> <td><i>OutBuf</i></td><td>Any transformed data is stored in <i>OutBuf</i>, converted to <code>ShapeCharset</code>.</td></tr> <tr> <td><i>Outsize</i></td><td>The number of bytes in <i>OutBuf</i>.</td></tr> <tr> <td><i>InpToOut</i></td><td>A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).</td></tr> <tr> <td><i>OutToInp</i></td><td>A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).</td></tr> </table>	<i>OutBuf</i>	Any transformed data is stored in <i>OutBuf</i> , converted to <code>ShapeCharset</code> .	<i>Outsize</i>	The number of bytes in <i>OutBuf</i> .	<i>InpToOut</i>	A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).	<i>OutToInp</i>	A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).
<i>OutBuf</i>	Any transformed data is stored in <i>OutBuf</i> , converted to <code>ShapeCharset</code> .								
<i>Outsize</i>	The number of bytes in <i>OutBuf</i> .								
<i>InpToOut</i>	A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).								
<i>OutToInp</i>	A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).								

Property

A weighted value that represents peculiar input string transformation properties with different connotations as explained below. If this argument is not a null pointer, it represents an array of values with the same number of elements as the source substring text before the transformation. Each byte will contain relevant “property” information of the corresponding element in *InpBuf* starting from the element pointed by *InpBufIndex*. The four rightmost bits of each “property” byte will contain information for bidirectional environments (when *ActiveDirectional* is *True*) and they will mean “*NestingLevels*.” The possible value from 0 to 15 represents the nesting level of the corresponding element in the *InpBuf* starting from the element pointed by *InpBufIndex*. If *ActiveDirectional* is false the content of *NestingLevel* bits will be ignored. The leftmost bit of each “property” byte will contain a “new cell indicator” for composed character environments, and will have a value of either 1 (for an element in *InpBuf* that is transformed to the beginning of a new cell) or 0 (for the “zero-length” composing character elements, when these are grouped into the same presentation cell with a non-composing character). Here again, each element of “property” pertains to the elements in the *InpBuf* starting from the element pointed by *InpBufIndex*. (Remember that this is not necessarily the beginning of *InpBuf*). If none of the transformation properties is required, the argument *Property* can be *NULL*. The use of “property” can be enhanced in the future to pertain to other possible usage in other environments.

The *InpBufIndex* argument is an offset value to the location of the transformed text. When *m_transform_layout()* is called, *InpBufIndex* contains the offset to the element in *InpBuf* that will be transformed first. (Note that this is not necessarily the first element in *InpBuf*). At the return from the transformation, *InpBufIndex* contains the offset to the first element in the *InpBuf* that has not been transformed. If the entire substring has been transformed successfully, *InpBufIndex* will be incremented by the amount defined by *InpSize*.

Each of these output arguments may be *NULL* to specify that no output is desired for the specific argument, but at least one of them should be set to a non-null value to perform any significant work.

The layout object maintains a directional state that keeps track of directional changes, based on the last segment transformed. The directional state is maintained across calls to the layout transformation functions and allows stream data to be processed with the layout functions. The directional state is reset to its initial state whenever any of the layout values *TypeOfText*, *Orientation*, or *ImplicitAlg* is modified by means of a call to *m_setvalues_layout()*.

The *layout_object* argument specifies a *LayoutObject* returned by the *m_create_layout()* function.

m_transform_layout(3LAYOUT)

The *OutBuf* argument contains the transformed data. This argument can be specified as a null pointer to indicate that no transformed data is required.

The encoding of the *OutBuf* argument depends on the *ShapeCharset* layout value defined in *layout_object*. If the *ActiveShapeEditing* layout value is not set (False), the encoding of *OutBuf* is guaranteed to be the same as the codeset of the locale associated with the *LayoutObject* defined by *layout_object*.

On input, the *OutSize* argument specifies the size of the output buffer in number of bytes. The output buffer should be large enough to contain the transformed result; otherwise, only a partial transformation is performed. If the *ActiveShapeEditing* layout value is set (True) the *OutBuf* should be allocated to contain at least the *InpSize* multiplied by *ShapeCharsetSize*.

On return, the *OutSize* argument is modified to the actual number of bytes placed in *OutBuf*.

When the *OutSize* argument is specified as zero, the function calculates the size of an output buffer large enough to contain the transformed text, and the result is returned in this field. The content of the buffers specified by *InpBuf* and *OutBuf*, and the value of *InpBufIndex*, remain unchanged. If *OutSize* = NULL, the EINVAL error condition should be returned.

If the *InpToOut* argument is not a null pointer, it points to an array of values with the same number of bytes in *InpBuf* starting with the one pointed by *InpBufIndex* and up to the end of the substring in the buffer. On output, the *n*th value in *InpToOut* corresponds to the *n*th byte in *InpBuf*. This value is the index (in units of bytes) in *OutBuf* that identifies the transformed *ShapeCharset* element of the *n*th byte in *InpBuf*. In the case of multibyte encoding, the index points (for each of the bytes of a code element in the *InpBuf*) to the first byte of the transformed code element in the *OutBuf*.

InpToOut may be specified as NULL if no index array from *InpBuf* to *OutBuf* is desired.

If the *OutToInp* argument is not a null pointer, it points to an array of values with the same number of bytes as contained in *OutBuf*. On output, the *n*th value in *OutToInp* corresponds to the *n*th byte in *OutBuf*. This value is the index in *InpBuf*, starting with the byte pointed to by *InpBufIndex*, that identifies the logical code element of the *n*th byte in *OutBuf*. In the case of multibyte encoding, the index will point for each of the bytes of a transformed code element in the *OutBuf* to the first byte of the code element in the *InpBuf*.

OutToInp may be specified as NULL if no index array from *OutBuf* to *InpBuf* is desired.

To perform shaping of a text string without reordering of code elements, the *layout_object* should be set with input and output layout value *TypeOfText* set to TEXT_VISUAL and both in and out of Orientation set to the same value.

RETURN VALUES If successful, the `m_transform_layout()` function returns 0. If unsuccessful, the returned value is -1 and the `errno` is set to indicate the source of error. When the size of *OutBuf* is not large enough to contain the entire transformed text, the input text state at the end of the uncompleted transformation is saved internally and the error condition E2BIG is returned in `errno`.

ERRORS The `m_transform_layout()` function may fail if:

E2BIG	The output buffer is full and the source text is not entirely processed.
EBADF	The layout values are set to a meaningless combination or the layout object is not valid.
EILSEQ	Transformation stopped due to an input code element that cannot be shaped or is invalid. The <i>InpBufIndex</i> argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the <i>ShapeCharset</i> layout value, or is an invalid code element not defined by the codeset of the locale of <i>layout_object</i> . The <code>mbtowc()</code> and <code>wctomb()</code> functions, when used in the same locale as the <i>LayoutObject</i> , can be used to determine if the code element is valid.
EINVAL	Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or <i>OutSize</i> contains NULL.
ERANGE	More than 15 embedding levels are in source text or <i>InpBuf</i> contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by <i>InpBufIndex</i> . An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes, the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer.

USAGE A *LayoutObject* will have a meaningful combination of default layout values. Whoever chooses to change the default layout values is responsible for making sure that the combination of layout values is meaningful. Otherwise, the result of `m_transform_layout()` might be unpredictable or implementation-specific with `errno` set to EBADF.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

m_transform_layout(3LAYOUT)

SEE ALSO attributes(5)

NAME	m_wtransform_layout – layout transformation for wide character strings								
SYNOPSIS	<pre>cc [flag...] file... -llayout [library...] #include <sys/layout.h> int m_wtransform_layout(LayoutObject layout_object, const wchar_t *InpBuf, const size_t InpSize, const void *OutBuf, size_t *Outsize, size_t *InpToOut, size_t *OutToInp, unsignedchar *Property, size_t *InpBufIndex);</pre>								
DESCRIPTION	The <code>m_wtransform_layout()</code> function performs layout transformations (reordering, shaping, cell determination) or provides additional information needed for layout transformation (such as the expected size of the transformed layout, the nesting level of different segments in the text and cross-references between the locations of the corresponding elements before and after the layout transformation). Both the input text and output text are wide character strings. The <code>m_wtransform_layout()</code> function transforms the input text in <i>InpBuf</i> according to the current layout values in <i>layout_object</i>. Any layout value whose value type is <code>LayoutTextDescriptor</code> describes the attributes of the <i>InpBuf</i> and <i>OutBuf</i> arguments. If the attributes are the same for both <i>InpBuf</i> and <i>OutBuf</i>, a null transformation is performed with respect to that specific layout value. The <i>InpBuf</i> argument specifies the source text to be processed. The <i>InpBuf</i> may not be NULL, unless there is a need to reset the internal state. The <i>InpSize</i> argument is the number of bytes within <i>InpBuf</i> to be processed by the transformation. Its value will not change after return from the transformation. <i>InpSize</i> set to -1 indicates that the text in <i>InpBuf</i> is delimited by a null code element. If <i>InpSize</i> is not set to -1, it is possible to have some null elements in the input buffer. This might be used, for example, for a “one shot” transformation of several strings, separated by nulls. Output of this function may be one or more of the following depending on the setting of the arguments: <table> <tr> <td><i>OutBuf</i></td><td>Any transformed data is stored in <i>OutBuf</i>, converted to <code>ShapeCharset</code>.</td></tr> <tr> <td><i>Outsize</i></td><td>The number of wide characters in <i>OutBuf</i>.</td></tr> <tr> <td><i>InpToOut</i></td><td>A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).</td></tr> <tr> <td><i>OutToInp</i></td><td>A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).</td></tr> </table>	<i>OutBuf</i>	Any transformed data is stored in <i>OutBuf</i> , converted to <code>ShapeCharset</code> .	<i>Outsize</i>	The number of wide characters in <i>OutBuf</i> .	<i>InpToOut</i>	A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).	<i>OutToInp</i>	A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).
<i>OutBuf</i>	Any transformed data is stored in <i>OutBuf</i> , converted to <code>ShapeCharset</code> .								
<i>Outsize</i>	The number of wide characters in <i>OutBuf</i> .								
<i>InpToOut</i>	A cross-reference from each <i>InpBuf</i> code element to the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).								
<i>OutToInp</i>	A cross-reference to each <i>InpBuf</i> code element from the transformed data. The cross-reference relates to the data in <i>InpBuf</i> starting with the first element that <i>InpBufIndex</i> points to (and not necessarily starting from the beginning of the <i>InpBuf</i>).								

m_wtransform_layout(3LAYOUT)

Property

A weighted value that represents peculiar input string transformation properties with different connotations as explained below. If this argument is not a nullpointer, it represents an array of values with the same number of elements as the source substring text before the transformation. Each byte will contain relevant “property” information of the corresponding element in *InpBuf* starting from the element pointed by *InpBufIndex*. The four rightmost bits of each “property” byte will contain information for bidirectional environments (when *ActiveDirectional* is True) and they will mean “*NestingLevels*.” The possible value from 0 to 15 represents the nesting level of the corresponding element in the *InpBuf* starting from the element pointed by *InpBufIndex*. If *ActiveDirectional* is false the content of *NestingLevel* bits will be ignored. The leftmost bit of each “property” byte will contain a “new cell indicator” for composed character environments, and will have a value of either 1 (for an element in *InpBuf* that is transformed to the beginning of a new cell) or 0 (for the “zero-length” composing character elements, when these are grouped into the same presentation cell with a non-composing character). Here again, each element of “property” pertains to the elements in the *InpBuf* starting from the element pointed by *InpBufIndex*. (Remember that this is not necessarily the beginning of *InpBuf*). If none of the transformation properties is required, the argument *Property* can be NULL. The use of “property” can be enhanced in the future to pertain to other possible usage in other environments.

The *InpBufIndex* argument is an offset value to the location of the transformed text. When *m_wtransform_layout()* is called, *InpBufIndex* contains the offset to the element in *InpBuf* that will be transformed first. (Note that this is not necessarily the first element in *InpBuf*). At the return from the transformation, *InpBufIndex* contains the offset to the first element in the *InpBuf* that has not been transformed. If the entire substring has been transformed successfully, *InpBufIndex* will be incremented by the amount defined by *InpSize*.

Each of these output arguments may be null to specify that no output is desired for the specific argument, but at least one of them should be set to a non-null value to perform any significant work.

In addition to the possible outputs above, *layout_object* maintains a directional state across calls to the transform functions. The directional state is reset to its initial state whenever any of the layout values *TypeOfText*, *Orientation*, or *ImplicitAlg* is modified by means of a call to *m_setvalues_layout()*.

The *layout_object* argument specifies a *LayoutObject* returned by the *m_create_layout()* function.

m_wtransform_layout(3LAYOUT)

The *OutBuf* argument contains the transformed data. This argument can be specified as a null pointer to indicate that no transformed data is required.

The encoding of the *OutBuf* argument depends on the ShapeCharset layout value defined in *layout_object*. If the ActiveShapeEditing layout value is not set (False), the encoding of *OutBuf* is guaranteed to be the same as the codeset of the locale associated with the LayoutObject defined by *layout_object*.

On input, the *OutSize* argument specifies the size of the output buffer in number of wide characters. The output buffer should be large enough to contain the transformed result; otherwise, only a partial transformation is performed. If the ActiveShapeEditing layout value is set (True) the *OutBuf* should be allocated to contain at least the *InpSize* multiplied by ShapeCharsetSize.

On return, the *OutSize* argument is modified to the actual number of code elements in *OutBuf*.

When the *OutSize* argument is specified as zero, the function calculates the size of an output buffer large enough to contain the transformed text, and the result is returned in this field. The content of the buffers specified by *InpBuf* and *OutBuf*, and the value of *InpBufIndex*, remain unchanged. If *OutSize* = NULL, the EINVAL error condition should be returned.

If the *InpToOut* argument is not a null pointer, it points to an array of values with the same number of wide characters in *InpBuf* starting with the one pointed by *InpBufIndex* and up to the end of the substring in the buffer. On output, the nth value in *InpToOut* corresponds to the nth byte in *InpBuf*. This value is the index (in units of wide characters) in *OutBuf* that identifies the transformed ShapeCharset element of the nth byte in *InpBuf*.

InpToOut may be specified as NULL if no index array from *InpBuf* to *OutBuf* is desired.

If the *OutToInp* argument is not a null pointer, it points to an array of values with the same number of wide characters as contained in *OutBuf*. On output, the nth value in *OutToInp* corresponds to the nth byte in *OutBuf*. This value is the index in *InpBuf*, starting with wide character byte pointed to by *InpBufIndex*, that identifies the logical code element of the nth wide character in *OutBuf*.

OutToInp may be specified as NULL if no index array from *OutBuf* to *InpBuf* is desired.

To perform shaping of a text string without reordering of code elements, the *layout_object* should be set with input and output layout value *TypeOfText* set to TEXT_VISUAL and both in and out of Orientation set to the same value.

RETURN VALUES

If successful, the m_wtransform_layout() function returns 0. If unsuccessful, the returned value is -1 and the errno is set to indicate the source of error. When the size of *OutBuf* is not large enough to contain the entire transformed text, the input text state at the end of the uncompleted transformation is saved internally and the error condition E2BIG is returned in errno.

m_wtransform_layout(3LAYOUT)

ERRORS	The <code>m_wtransform_layout()</code> function may fail if: <table> <tr> <td data-bbox="443 359 521 380">E2BIG</td><td data-bbox="662 359 1312 415">The output buffer is full and the source text is not entirely processed.</td></tr> <tr> <td data-bbox="443 436 521 457">EBADF</td><td data-bbox="662 436 1354 493">The layout values are set to a meaningless combination or the layout object is not valid.</td></tr> <tr> <td data-bbox="443 514 537 535">EILSEQ</td><td data-bbox="662 514 1409 781">Transformation stopped due to an input code element that cannot be shaped or is invalid. The <i>InpBufIndex</i> argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the <i>ShapeCharset</i> layout value, or is an invalid code element not defined by the codeset of the locale of <i>layout_object</i>. The <code>mbtowc()</code> and <code>wctomb()</code> functions, when used in the same locale as the <i>LayoutObject</i>, can be used to determine if the code element is valid.</td></tr> <tr> <td data-bbox="443 802 537 823">EINVAL</td><td data-bbox="662 802 1406 858">Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or <i>OutSize</i> contains <code>NULL</code>.</td></tr> <tr> <td data-bbox="443 879 537 900">ERANGE</td><td data-bbox="662 879 1409 1178"> More than 15 embedding levels are in source text or <i>InpBuf</i> contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by <i>InpBufIndex</i>. An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer. </td></tr> </table>	E2BIG	The output buffer is full and the source text is not entirely processed.	EBADF	The layout values are set to a meaningless combination or the layout object is not valid.	EILSEQ	Transformation stopped due to an input code element that cannot be shaped or is invalid. The <i>InpBufIndex</i> argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the <i>ShapeCharset</i> layout value, or is an invalid code element not defined by the codeset of the locale of <i>layout_object</i> . The <code>mbtowc()</code> and <code>wctomb()</code> functions, when used in the same locale as the <i>LayoutObject</i> , can be used to determine if the code element is valid.	EINVAL	Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or <i>OutSize</i> contains <code>NULL</code> .	ERANGE	More than 15 embedding levels are in source text or <i>InpBuf</i> contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by <i>InpBufIndex</i>. An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer.
E2BIG	The output buffer is full and the source text is not entirely processed.										
EBADF	The layout values are set to a meaningless combination or the layout object is not valid.										
EILSEQ	Transformation stopped due to an input code element that cannot be shaped or is invalid. The <i>InpBufIndex</i> argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the <i>ShapeCharset</i> layout value, or is an invalid code element not defined by the codeset of the locale of <i>layout_object</i> . The <code>mbtowc()</code> and <code>wctomb()</code> functions, when used in the same locale as the <i>LayoutObject</i> , can be used to determine if the code element is valid.										
EINVAL	Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or <i>OutSize</i> contains <code>NULL</code> .										
ERANGE	More than 15 embedding levels are in source text or <i>InpBuf</i> contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by <i>InpBufIndex</i>. An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer.										
USAGE	A <i>LayoutObject</i> will have a meaningful combination of default layout values. Whoever chooses to change the default layout values is responsible for making sure that the combination of layout values is meaningful. Otherwise, the result of <code>m_wtransform_layout()</code> might be unpredictable or implementation-specific with <code>errno</code> set to <code>EBADF</code>.										
EXAMPLES	EXAMPLE 1 Shaping and reordering input string into output buffer The following example illustrated what the different arguments of <code>m_wtransform_layout()</code> look like when a string in <i>InpBuf</i> is shaped and reordered into <i>OutBuf</i>. Upper-case letters in the example represent left-to-right letters while lower-case letters represent right-to-left letters. <i>xyz</i> represents the shapes of <i>cde</i>. <table> <tr> <td data-bbox="443 1570 553 1591">Position:</td><td data-bbox="792 1570 911 1591">0123456789</td></tr> <tr> <td data-bbox="443 1596 521 1617">InpBuf:</td><td data-bbox="792 1596 911 1617">AB cde 12z</td></tr> <tr> <td data-bbox="443 1648 553 1669">Position:</td><td data-bbox="792 1648 911 1669">0123456789</td></tr> <tr> <td data-bbox="443 1673 521 1694">OutBuf:</td><td data-bbox="792 1673 911 1694">AB 12 zyxZ</td></tr> <tr> <td data-bbox="443 1726 553 1747">Position:</td><td data-bbox="792 1726 911 1747">0123456789</td></tr> </table>	Position:	0123456789	InpBuf:	AB cde 12z	Position:	0123456789	OutBuf:	AB 12 zyxZ	Position:	0123456789
Position:	0123456789										
InpBuf:	AB cde 12z										
Position:	0123456789										
OutBuf:	AB 12 zyxZ										
Position:	0123456789										

EXAMPLE 1 Shaping and reordering input string into output buffer (Continued)

```
OutToInp:                0127865439

Position:                 0123456789
Property.NestLevel:       0001111220
Property.CelBdry:         1111111111
```

The values (encoded in binary) returned in the *Property* argument define the directionality of each code element in the source text as defined by the type of algorithm used within the *layout_object*. While the algorithm may be implementation dependent, the resulting values and levels are defined such as to allow a single method to be used in determining the directionality of the source text. The base rules are:

- Odd levels are always RTL.
- Even levels are always LTR.
- The *Orientation* layout value setting determines the initial level (0 or 1) used.

Within a *Property* array each increment in the level indicates the corresponding code elements should be presented in the opposite direction. Callers of this function should realize that the *Property* values for certain code elements is dependent on the context of the given character and the layout values: *Orientation* and *ImplicitAlg*. Callers should not assume that a given code element always has the same *Property* value in all cases.

EXAMPLE 2 Algorithm to handle nesting

The following is an example of a standard presentation algorithm that handles nesting correctly. The goal of this algorithm is ultimately to return to a zero nest level. Note that more efficient algorithms do exist; the following is provided for clarity rather than for efficiency.

1. Search for the highest next level in the string.
2. Reverse all surrounding code elements of the same level. Reduce the nest level of these code elements by 1.
3. Repeat 1 and 2 until all code elements are of level 0.

The following shows the progression of the example from above:

Position:	0123456789	0123456789	0123456789
InpBuf:	AB cde 12Z	AB cde 21Z	AB 12 edcZ
Property.NestLevel:	0001111220	0001111110	0000000000
Property.CellBdry:	1111111111	1111111111	1111111111

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

m_wtransform_layout(3LAYOUT)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO attributes(5)

NAME	newDmiOctetString – create DmiOctetString in dynamic memory				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <dmi/util.hh> DmiOctetString_t *newDmiOctetString(DmiOctetString_t *<i>str</i>);</pre>				
DESCRIPTION	The newDmiOctetString() function creates a DmiOctetString in dynamic memory and returns a pointer to the newly created DmiOctetString. The function returns NULL if no memory is available.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-level	MT-Safe				
SEE ALSO	libdmi(3LIB), attributes(5)				

newDmiString(3DMI)

NAME	newDmiString – create DmiString in dynamic memory
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <dmi/util.hh> DmiString_t *newDmiString(char *str);</pre>
DESCRIPTION	The newDmiString() function creates a DmiString in dynamic memory and returns a pointer to the newly created DmiString. The function returns NULL if no memory is available.
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	MT-Safe

SEE ALSO freeDmiString(3DMI), libdmi(3LIB), attributes(5)

NAME	nextafter – next representable double-precision floating-point number				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double nextafter(double <i>x</i>, double <i>y</i>);</pre>				
DESCRIPTION	The nextafter() function computes the next representable double-precision floating-point value following <i>x</i> in the direction of <i>y</i> . Thus, if <i>y</i> is less than <i>x</i> , nextafter() returns the largest representable floating-point number less than <i>x</i> .				
RETURN VALUES	The nextafter() function returns the next representable double-precision floating-point value following <i>x</i> in the direction of <i>y</i>. If <i>x</i> or <i>y</i> is NaN, then nextafter() returns NaN. If <i>x</i> is finite and the correct function value would overflow, nextafter() returns ±HUGE_VAL (according to the sign of <i>x</i>) and sets errno to ERANGE.				
ERRORS	The nextafter() function will fail if: ERANGE The correct value would overflow.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	attributes(5),				

nlist(3ELF)

NAME	nlist – get entries from name list						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file</i> ... -lelf [<i>library</i> ...] #include <nlist.h> int nlist(const char *<i>filename</i>, struct nlist *<i>nl</i>);</pre>						
DESCRIPTION	nlist() examines the name list in the executable file whose name is pointed to by <i>filename</i>, and selectively extracts a list of values and puts them in the array of structures pointed to by <i>nl</i>. The name list <i>nl</i> consists of an array of structures containing names of variables, types, and values. The list is terminated with a null name, that is, a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type, value, storage class, and section number of the name are inserted in the other fields. The type field may be set to 0 if the file was not compiled with the <i>-g</i> option to cc(1B). nlist() will always return the information for an external symbol of a given name if the name exists in the file. If an external symbol does not exist, and there is more than one symbol with the specified name in the file (such as static symbols defined in separate files), the values returned will be for the last occurrence of that name in the file. If the name is not found, all fields in the structure except <i>n_name</i> are set to 0. This function is useful for examining the system name list kept in the file <i>/dev/ksyms</i>. In this way programs can obtain system addresses that are up to date.						
RETURN VALUES	All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list. nlist() returns 0 on success, -1 on error.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	Safe						
SEE ALSO	cc(1B), elf(3ELF), kvm_nlist(3KVM), kvm_open(3KVM), libelf(3LIB), a.out(4), attributes(5), ksyms(7D), mem(7D)						

NAME	NOTE, _NOTE – annotate source code with info for tools
SYNOPSIS	<pre>#include <note.h> NOTE (NoteInfo) ; or #include<sys/note.h> _NOTE (NoteInfo) ;</pre>
DESCRIPTION	These macros are used to embed information for tools in program source. A use of one of these macros is called an “annotation”. A tool may define a set of such annotations which can then be used to provide the tool with information that would otherwise be unavailable from the source code. Annotations should, in general, provide documentation useful to the human reader. If information is of no use to a human trying to understand the code but is necessary for proper operation of a tool, use another mechanism for conveying that information to the tool (one which does not involve adding to the source code), so as not to detract from the readability of the source. The following is an example of an annotation which provides information of use to a tool and to the human reader (in this case, which data are protected by a particular lock, an annotation defined by the static lock analysis tool <code>lock_lint</code>). <pre>NOTE (MUTEX_PROTECTS_DATA(foo_lock, foo_list Foo))</pre> Such annotations do not represent executable code; they are neither statements nor declarations. They should not be followed by a semicolon. If a compiler or tool that analyzes C source does not understand this annotation scheme, then the tool will ignore the annotations. (For such tools, <code>NOTE(x)</code> expands to nothing.) Annotations may only be placed at particular places in the source. These places are where the following C constructs would be allowed: ■ a top-level declaration (that is, a declaration not within a function or other construct) ■ a declaration or statement within a block (including the block which defines a function) ■ a member of a struct or union. Annotations are not allowed in any other place. For example, the following are illegal: <pre>x = y + NOTE(...) z ; typedef NOTE(...) unsigned int uint ;</pre> While <code>NOTE</code> and <code>_NOTE</code> may be used in the places described above, a particular type of annotation may only be allowed in a subset of those places. For example, a particular annotation may not be allowed inside a struct or union definition.

NOTE(3EXT)

NOTE vs _NOTE

Ordinarily, NOTE should be used rather than _NOTE, since use of _NOTE technically makes a program non-portable. However, it may be inconvenient to use NOTE for this purpose in existing code if NOTE is already heavily used for another purpose. In this case one should use a different macro and write a header file similar to /usr/include/note.h which maps that macro to _NOTE in the same manner. For example, the following makes FOO such a macro:

```
#ifndef _FOO_H
#define _FOO_H
#define FOO _NOTE
#include <sys/note.h>
#endif
```

Public header files which span projects should use _NOTE rather than NOTE, since NOTE may already be used by a program which needs to include such a header file.

NoteInfo Argument

The actual *NoteInfo* used in an annotation should be specified by a tool that deals with program source (see the documentation for the tool to determine which annotations, if any, it understands).

NoteInfo must have one of the following forms:

NoteName
NoteName (Args)

where *NoteName* is simply an identifier which indicates the type of annotation, and *Args* is something defined by the tool that specifies the particular *NoteName*. The general restrictions on *Args* are that it be compatible with an ANSI C tokenizer and that unquoted parentheses be balanced (so that the end of the annotation can be determined without intimate knowledge of any particular annotation).

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

SEE ALSO

note(4), attributes(5)

NAME	nvlist_add_boolean, nvlist_add_byte, nvlist_add_int16, nvlist_add_uint16, nvlist_add_int32, nvlist_add_uint32, nvlist_add_int64, nvlist_add_uint64, nvlist_add_string, nvlist_add_byte_array, nvlist_add_int16_array, nvlist_add_uint16_array, nvlist_add_int32_array, nvlist_add_uint32_array, nvlist_add_int64_array, nvlist_add_uint64_array, nvlist_add_string_array – add new name-value pair to nvlist_t
SYNOPSIS	<pre>cc [flag...] file... -lnvpair [library...] #include <libnvpair.h> int nvlist_add_boolean(nvlist_t *nvl, char *name); int nvlist_add_byte(nvlist_t *nvl, char *name, uchar_t val); int nvlist_add_int16(nvlist_t *nvl, char *name, int16_t val); int nvlist_add_uint16(nvlist_t *nvl, char *name, uint16_t val); int nvlist_add_int32(nvlist_t *nvl, char *name, int32_t val); int nvlist_add_uint32(nvlist_t *nvl, char *name, uint32_t val); int nvlist_add_int64(nvlist_t *nvl, char *name, int64_t val); int nvlist_add_uint64(nvlist_t *nvl, char *name, uint64_t val); int nvlist_add_string(nvlist_t *nvl, char *name, char *val); int nvlist_add_byte_array(nvlist_t *nvl, char *name, uchar_t *val, uint_t nelem); int nvlist_add_int16_array(nvlist_t *nvl, char *name, int16_t *val, uint_t nelem); int nvlist_add_uint16_array(nvlist_t *nvl, char *name, uint16_t *val, uint_t nelem); int nvlist_add_int32_array(nvlist_t *nvl, char *name, int32_t *val, uint_t nelem); int nvlist_add_uint32_array(nvlist_t *nvl, char *name, uint32_t *val, uint_t nelem); int nvlist_add_int64_array(nvlist_t *nvl, char *name, int64_t *val, uint_t nelem); int nvlist_add_uint64_array(nvlist_t *nvl, char *name, uint64_t *val, uint_t nelem); int nvlist_add_string_array(nvlist_t *nvl, char *name, char **val, uint_t nelem);</pre>
PARAMETERS	<i>nvl</i> The nvlist_t (name-value pair list) to be processed. <i>name</i> Name of the nvpair (name-value pair). <i>nelem</i> Number of elements in value (that is, array size).

nvlist_add_boolean(3NVP AIR)

	<i>val</i> Value or starting address of the array value.						
DESCRIPTION	These functions add a new name-value pair to an <code>nvlist_t</code>. The uniqueness of <code>nvpair</code> name and data types follows the <i>nvflag</i> argument specified for <code>nvlist_alloc()</code>. See <code>nvlist_alloc(3NVP AIR)</code>. If <code>NV_UNIQUE_NAME</code> was specified for <i>nvflag</i>, existing <code>nvpairs</code> with matching names are removed before the new <code>nvpair</code> is added. If <code>NV_UNIQUE_NAME_TYPE</code> was specified for <i>nvflag</i>, existing <code>nvpairs</code> with matching names and data types are removed before the new <code>nvpair</code> is added. If neither was specified for <i>nvflag</i>, the new <code>nvpair</code> is unconditionally added at the end of the list. The library preserves the order of the name-value pairs across packing, unpacking, and duplication.						
RETURN VALUES	These functions return 0 on success and an error value on failure.						
ERRORS	These functions will fail if: <code>EINVAL</code> There is an invalid argument. <code>ENOMEM</code> There is insufficient memory.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>libnvpair(3NVP AIR)</code> , <code>attributes(5)</code>						

NAME	nvlist_alloc, nvlist_free, nvlist_size, nvlist_pack, nvlist_unpack, nvlist_dup – manage a name-value pair list						
SYNOPSIS	<pre>cc [flag...] file... -lnvpair [library...] #include <libnvpair.h> int nvlist_alloc(nvlist_t **nvp, uint_t nvflag, int flag); void nvlist_free(nvlist_t *nvl); int nvlist_size(nvlist_t *nvl, size_t *size, int encoding); int nvlist_pack(nvlist_t *nvl, char **bufp, size_t *buflen, int encoding, int flag); int nvlist_unpack(char *buf, size_t buflen, nvlist_t **nvp, int flag); int nvlist_dup(nvlist_t *nvl, nvlist_t **nvp, int flag);</pre>						
PARAMETERS	nvp	Address of a pointer to nvlist_t.					
	nvflag	Specify bit fields defining nvlist properties:					
		NV_UNIQUE_NAME	The nvpair names are unique.				
		NV_UNIQUE_NAME_TYPE	Name-data type combination is unique				
	flag	Specify 0. Reserved for future use.					
	nvl	The nvlist_t to be processed.					
	size	Pointer to buffer to contain the encoded size.					
	bufp	Address of buffer to pack nvlist into. Must be 8-byte aligned. If NULL, library will allocate memory.					
	buf	Buffer containing packed nvlist.					
	buflen	Size of buffer bufp or buf points to.					
	encoding	Encoding method for packing.					
DESCRIPTION	The nvlist_alloc() function allocates a new name-value pair list and updates nvp to point to the handle. The argument nvflag specifies nvlist properties to remain persistent across packing, unpacking, and duplication. The nvlist_free() function frees a name-value pair list. The nvlist_size() function returns the minimum size of a contiguous buffer large enough to pack nvl. The encoding parameter specifies the method of encoding when packing nvl. Supported encoding methods are: <table><tr><td>NV_ENCODE_NATIVE</td><td>Straight bcopy() as described in bcopy(3C).</td></tr><tr><td>NV_ENCODE_XDR</td><td>Use XDR encoding, suitable for sending to another host.</td></tr></table>			NV_ENCODE_NATIVE	Straight bcopy() as described in bcopy(3C).	NV_ENCODE_XDR	Use XDR encoding, suitable for sending to another host.
NV_ENCODE_NATIVE	Straight bcopy() as described in bcopy(3C).						
NV_ENCODE_XDR	Use XDR encoding, suitable for sending to another host.						

`nvlist_alloc(3NVPAIR)`

The `nvlist_pack()` function packs *nvlist* into contiguous memory starting at **bufp*. The *encoding* parameter specifies the method of encoding (see above).

- If **bufp* is not NULL, **bufp* is expected to be a caller-allocated buffer of size **buflen*.
- If **bufp* is NULL, the library will allocate memory and update **bufp* to point to the memory and update **buflen* to contain the size of the allocated memory.

The `nvlist_unpack()` function takes a buffer with a packed `nvlist_t` and unpacks it into a searchable `nvlist_t`. The library allocates memory for `nvlist_t`. The caller is responsible for freeing the memory by calling `nvlist_free()`.

The `nvlist_dup()` function makes a copy of *nvlist* and updates *nvlistp* to point to the copy.

RETURN VALUES These functions return 0 on success and an error value on failure.

ERRORS These functions will fail if:

`EINVAL` There is an invalid argument.

The `nvlist_alloc()`, `nvlist_dup()`, `nvlist_pack()`, and `nvlist_unpack()` functions will fail if:

`ENOMEM` There is insufficient memory.

The `nvlist_pack()` and `nvlist_unpack()` functions will fail if:

`EFAULT` An encode/decode error occurs.

`ENOTSUP` An encode/decode method is not supported.

EXAMPLES

```
/*
 * Program to read or create an nvlist.
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <libnvpair.h>

/* generate a packed nvlist */
static int
create_packed_nvlist(char **buf, uint_t *buflen, int encode)
{
    uchar_t bytes[] = {0xaa, 0xbb, 0xcc, 0xdd};
    int16_t int16[] = {0, 1, 2};
    int32_t int32[] = {3, 4, 5};
    uint64_t uint64[] = {0x100000007, 0x100000008, 0x100000009};
    char *strs[] = {"child0", "child1", "child2"};
    int err;
    nvlist_t *nvl;

    err = nvlist_alloc(&nvl, NV_UNIQUE_NAME, 0);    /* allocate list */
```

```

    if (err) {
        (void) printf("nvlist_alloc() failed\n");
        return (err);
    }

    /* add a value of each type */
    if ((nvlist_add_boolean(nvl, "bool") != 0) ||
        (nvlist_add_byte(nvl, "byte", bytes[0]) != 0) ||
        (nvlist_add_int16(nvl, "int16", int16[0]) != 0) ||
        (nvlist_add_int32(nvl, "int32", int32[0]) != 0) ||
        (nvlist_add_uint64(nvl, "uint64", uint64[0]) != 0) ||
        (nvlist_add_string(nvl, "string", str[0]) != 0) ||
        (nvlist_add_byte_array(nvl, "byte_array", bytes, 4) != 0) ||
        (nvlist_add_int16_array(nvl, "int16_array", int16, 3) != 0) ||
        (nvlist_add_int32_array(nvl, "int32_array", int32, 3) != 0) ||
        (nvlist_add_uint64_array(nvl, "uint64_array", uint64, 3) != 0) ||
        (nvlist_add_string_array(nvl, "string_array", str, 3) != 0)) {
        nvlist_free(nvl);
        return (-1);
    }

    err = nvlist_size(nvl, buflen, encode);
    if (err) {
        (void) printf("nvlist_size: %s\n", strerror(err));
        return (err);
    }

    /* pack into contig. memory */
    err = nvlist_pack(nvl, buf, buflen, encode, 0);
    if (err)
        (void) printf("nvlist_pack: %s\n", strerror(err));

    /* free the original list */
    nvlist_free(nvl);
    return (err);
}

/* read a packed nvlist from file or create a packed nvlist */
static int
get_nvlist_buf(char *file, char **buf, size_t *buflen) {
    int fd, rv;
    struct stat sbuf;

    if (file == NULL)
        return (create_packed_nvlist(buf, buflen, NV_ENCODE_NATIVE));

    /* read from file */
    fd = open(file, O_RDONLY);
    if (fd == -1) {
        (void) printf("cannot open file %s\n", file);
        return (-1);
    }
    (void) fstat(fd, &sbuf);
    *buflen = sbuf.st_size;
    *buf = malloc(*buflen);
    if (*buf == NULL) {

```

nvlist_alloc(3NVPAIR)

```

        (void) printf("out of memory\n");
        return (-1);
    }
    rv = read(fd, *buf, *buflen);
    (void) close(fd);
    return (rv);
}

/* selectively print nvpairs */
static void
nvlist_lookup_and_print(nvlist_t *nvl)
{
    char **str_val;
    int i, int_val;
    uint_t nval;

    if (nvlist_lookup_int32(nvl, "int32", &int_val) == 0)
        (void) printf("int32 = %d\n", int_val);
    if (nvlist_lookup_string_array(nvl, "string_array", &str_val, &nval)
        == 0) {
        (void) printf("string_array =");
        for (i = 0; i < nval; i++)
            (void) printf(" %s", str_val[i]);
        (void) printf("\n");
    }
}

void
main(int argc, char *argv[])
{
    int c, err;
    char *file = NULL, *buf = NULL;
    size_t buflen;
    nvlist_t *nvl = NULL;

    while ((c = getopt(argc, argv, "r:")) != EOF)
        switch (c) {
            case 'r':
                file = optarg;
                break;
            default:
                (void) printf("Usage: %s [ -r file ]", argv[0]);
                return;
        }

    if (get_nvlist_buf(file, &buf, &buflen) != 0) {
        (void) printf("cannot get packed nvlist buffer\n");
        return;
    }

    /* unpack into an nvlist_t */
    err = nvlist_unpack(buf, buflen, &nvl, 0);
    if (err) {
        (void) printf("nvlist_unpack(): %s\n", strerror(err));
        return;
    }
}

```

nvlist_alloc(3NVPAIR)

```
/* selectively print out attributes */  
nvlist_lookup_and_print(nvl);  
return;  
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libnvpair(3NVPAIR), attributes(5)

nvlist_lookup_boolean(3NVP AIR)

NAME	nvlist_lookup_boolean, nvlist_lookup_byte, nvlist_lookup_int16, nvlist_lookup_uint16, nvlist_lookup_int32, nvlist_lookup_uint32, nvlist_lookup_int64, nvlist_lookup_uint64, nvlist_lookup_string, nvlist_lookup_byte_array, nvlist_lookup_int16_array, nvlist_lookup_uint16_array, nvlist_lookup_int32_array, nvlist_lookup_uint32_array, nvlist_lookup_int64_array, nvlist_lookup_uint64_array, nvlist_lookup_string_array – match name and type indicated by the interface name and retrieve data value
SYNOPSIS	<pre>cc [flag...] file... -lnvpair [library...] #include <libnvpair.h> int nvlist_lookup_boolean(nvlist_t *nvl, char *name); int nvlist_lookup_byte(nvlist_t *nvl, char *name, uchar_t *val); int nvlist_lookup_int16(nvlist_t *nvl, char *name, int16_t *val); int nvlist_lookup_uint16(nvlist_t *nvl, char *name, uint16_t *val); int nvlist_lookup_int32(nvlist_t *nvl, char *name, int32_t *val); int nvlist_lookup_uint32(nvlist_t *nvl, char *name, uint32_t *val); int nvlist_lookup_int64(nvlist_t *nvl, char *name, int64_t *val); int nvlist_lookup_uint64(nvlist_t *nvl, char *name, uint64_t *val); int nvlist_lookup_string(nvlist_t *nvl, char *name, char **val); int nvlist_lookup_byte_array(nvlist_t *nvl, char *name, uchar_t **val, uint_t *nelem); int nvlist_lookup_int16_array(nvlist_t *nvl, char *name, int16_t **val, uint_t *nelem); int nvlist_lookup_uint16_array(nvlist_t *nvl, char *name, uint16_t **val, uint_t *nelem); int nvlist_lookup_int32_array(nvlist_t *nvl, char *name, int32_t **val, uint_t *nelem); int nvlist_lookup_uint32_array(nvlist_t *nvl, char *name, uint32_t **val, uint_t *nelem); int nvlist_lookup_int64_array(nvlist_t *nvl, char *name, int64_t **val, uint_t *nelem); int nvlist_lookup_uint64_array(nvlist_t *nvl, char *name, uint64_t **val, uint_t *nelem); int nvlist_lookup_string_array(nvlist_t *nvl, char *name, char ***val, uint_t *nelem);</pre>
PARAMETERS	<i>nvl</i> The nvlist_t to be processed. <i>name</i> Name of the name-value pair to search. <i>nelem</i> Address to store the number of elements in value.

nvlist_lookup_boolean(3NVPAIR)

	<i>val</i>	Address to store the starting address of the value.						
DESCRIPTION	These functions find the <code>nvpair</code> (name-value pair) that matches the name and type as indicated by the interface name. If one is found, <i>nelem</i> and <i>val</i> are modified to contain the number of elements in value and the starting address of data, respectively. These functions work for <code>nvlists</code> (lists of name-value pairs) allocated with <code>NV_UNIQUE_NAME</code> or <code>NV_UNIQUE_NAME_TYPE</code> specified in <code>nvlist_alloc()</code>. (See <code>nv_list_alloc(3NVPAIR)</code>.) If this is not the case, the function returns <code>ENOTSUP</code> because the list potentially contains multiple <code>nvpairs</code> with the same name and type. All memory required for storing the array elements, including string value, are managed by the library. References to such data remain valid until <code>nvlist_free()</code> is called on <i>nvlist</i>.							
RETURN VALUES	These functions return 0 on success and an error value on failure.							
ERRORS	These functions will fail if: <table><tr><td><code>EINVAL</code></td><td>There is an invalid argument.</td></tr><tr><td><code>ENOENT</code></td><td>No matching name-value pair is found</td></tr><tr><td><code>ENOTSUP</code></td><td>An encode/decode method is not supported.</td></tr></table>		<code>EINVAL</code>	There is an invalid argument.	<code>ENOENT</code>	No matching name-value pair is found	<code>ENOTSUP</code>	An encode/decode method is not supported.
<code>EINVAL</code>	There is an invalid argument.							
<code>ENOENT</code>	No matching name-value pair is found							
<code>ENOTSUP</code>	An encode/decode method is not supported.							
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Interface Stability	Evolving							
MT-Level	MT-Safe							
SEE ALSO	<code>libnvpair(3NVPAIR)</code> , <code>attributes(5)</code>							

nvlist_next_nvpair(3NVPAR)

NAME	nvlist_next_nvpair, nvpair_name, nvpair_type – return data regarding name-value pairs
SYNOPSIS	<pre>cc [flag...] file... -lnvpair [library...] #include <libnvpair.h> nvpair_t *nvlist_next_nvpair(nvlist_t *nvl, nvpair_t *nvpair); char *nvpair_name(nvpair_t *nvpair); data_type_t nvpair_type(nvpair_t *nvpair);</pre>
PARAMETERS	<i>nvl</i> The nvlist_t to be processed. <i>nvpair</i> Handle to a name-value pair.
DESCRIPTION	The nvlist_next_nvpair() function returns a handle to the next nvpair in the list following nvpair. If nvpair is NULL, the first pair is returned. If nvpair is the last pair in the nvlist, NULL is returned. The nvpair_name() function returns a string containing the name of nvpair. The nvpair_type() function retrieves the value of the nvpair in the form of enumerated type data_type_t. This is used to determine the appropriate nvpair_*() function to call for retrieving the value.
RETURN VALUES	Upon successful completion, nvpair_name() returns a string containing the name of the name-value pair. Upon successful completion, nvpair_type() returns an enumerated data type data_type_t. Possible values for data_type_t are as follows: <pre>DATA_TYPE_BOOLEAN DATA_TYPE_BYTE DATA_TYPE_INT16 DATA_TYPE_UINT16 DATA_TYPE_INT32 DATA_TYPE_UINT32 DATA_TYPE_INT64 DATA_TYPE_UINT64 DATA_TYPE_STRING DATA_TYPE_BYTE_ARRAY DATA_TYPE_INT16_ARRAY DATA_TYPE_UINT16_ARRAY DATA_TYPE_INT32_ARRAY DATA_TYPE_UINT32_ARRAY DATA_TYPE_INT64_ARRAY DATA_TYPE_UINT64_ARRAY DATA_TYPE_STRING_ARRAY</pre> Upon reaching the end of a list, nvlist_next_pair() returns NULL. Otherwise, the function returns a handle to next nvpair in the list.
ERRORS	No errors are defined.

`nvlist_next_nvpair(3NVPAIR)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `libnvpair(3NVPAIR)`, `attributes(5)`

NOTES The enumerated nvpair data types might not be an exhaustive list and new data types can be added. An application using the data type enumeration, `data_type_t`, should be written to expect or ignore new data types.

nvlist_remove(3NVPAIR)

NAME	nvlist_remove, nvlist_remove_all – remove name-value pairs						
SYNOPSIS	<pre>cc [flag ...] file ...-lnvpair [library ...] #include <libnvpair.h> void nvlist_remove(nvlist_t *nvl, char *name, data_type_t type); void nvlist_remove_all(nvlist_t *nvl, char *name);</pre>						
PARAMETERS	<i>nvl</i> The nvlist_t to be processed. <i>name</i> Name of the name-value pair to be removed. <i>type</i> Data type of the nvpair to be removed.						
DESCRIPTION	The nvlist_remove() function removes the first occurrence of nvpair that matches the name and the type. The nvlist_remove_all() function removes all occurrences of nvpair that match the name, regardless of type.						
RETURN VALUES	No return values are defined.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	libnvpair(3NVPAIR), attributes(5)						

nvpair_value_byte(3NVP-PAIR)

NAME	nvpair_value_byte, nvpair_value_int16, nvpair_value_uint16, nvpair_value_int32, nvpair_value_uint32, nvpair_value_int64, nvpair_value_uint64, nvpair_value_string, nvpair_value_byte_array, nvpair_value_int16_array, nvpair_value_uint16_array, nvpair_value_int32_array, nvpair_value_uint32_array, nvpair_value_int64_array, nvpair_value_uint64_array, nvpair_value_string_array – retrieve value from a name-value pair
SYNOPSIS	<pre>cc [flag...] file... -lnvpair [library...] #include <libnvpair.h> int nvpair_value_byte(nvpair_t *nvpair, uchar_t *val); int nvpair_value_int16(nvpair_t *nvpair, int16_t *val); int nvpair_value_uint16(nvpair_t *nvpair, uint16_t *val); int nvpair_value_int32(nvpair_t *nvpair, int32_t *val); int nvpair_value_uint32(nvpair_t *nvpair, uint32_t *val); int nvpair_value_int64(nvpair_t *nvpair, int64_t *val); int nvpair_value_uint64(nvpair_t *nvpair, uint64_t *val); int nvpair_value_string(nvpair_t *nvpair, char **val); int nvpair_value_byte_array(nvpair_t *nvpair, uchar_t **val, uint_t *nelem); int nvpair_value_int16_array(nvpair_t *nvpair, int16_t **val, uint_t *nelem); int nvpair_value_uint16_array(nvpair_t *nvpair, uint16_t **val, uint_t *nelem); int nvpair_value_int32_array(nvpair_t *nvpair, int32_t **val, uint_t *nelem); int nvpair_value_uint32_array(nvpair_t *nvpair, uint32_t **val, uint_t *nelem); int nvpair_value_int64_array(nvpair_t *nvpair, int64_t **val, uint_t *nelem); int nvpair_value_uint64_array(nvpair_t *nvpair, uint64_t **val, uint_t *nelem); int nvpair_value_string_array(nvpair_t *nvpair, char ***val, uint_t *nelem);</pre>
PARAMETERS	<i>nvpair</i> Name-value pair to be processed. <i>nelem</i> Address to store the number of elements in value. <i>val</i> Address to store the value or the starting address of the array value.

`nvpair_value_byte(3NVPAIR)`

DESCRIPTION

These functions retrieve the value of *nvpair*. The data type of *nvpair* must match the interface name for the call to be successful.

There is no `nvpair_value_boolean()`; the existence of the name implies the value is true.

For array data types, including string, the memory containing the data is managed by the library and references to the value remains valid until `nvlist_free()` is called on the `nvlist_t` from which *nvpair* is obtained. See `nvlist_free(3NVPAIR)`.

RETURN VALUES

These functions return 0 on success and an error value on failure.

ERRORS

These functions will fail if:

EINVAL Either one of the arguments is NULL or the type of *nvpair* does not match the function name.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

`libnvpair(3LIB)`, `attributes(5)`

NAME	p2open, p2close – open, close pipes to and from a command
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int p2open(const char *<i>cmd</i>, FILE *<i>fp</i>[2]); int p2close(FILE *<i>fp</i>[2]);</pre>
DESCRIPTION	p2open() forks and execs a shell running the command line pointed to by <i>cmd</i>. On return, <i>fp</i>[0] points to a FILE pointer to write the command's standard input and <i>fp</i>[1] points to a FILE pointer to read from the command's standard output. In this way the program has control over the input and output of the command. The function returns 0 if successful; otherwise, it returns -1. p2close() is used to close the file pointers that p2open() opened. It waits for the process to terminate and returns the process status. It returns 0 if successful; otherwise, it returns -1.
RETURN VALUES	A common problem is having too few file descriptors. p2close() returns -1 if the two file pointers are not from the same p2open().
EXAMPLES	EXAMPLE 1 Example of file descriptors. <pre>#include <stdio.h> #include <libgen.h> main(argc,argv) int argc; char **argv; { FILE *fp[2]; pid_t pid; char buf[16]; pid=p2open("/usr/bin/cat", fp); if (pid == -1) { fprintf(stderr, "p2open failed\n"); exit(1); } write(fileno(fp[0]),"This is a test\n", 16); if(read(fileno(fp[1]), buf, 16) <=0) fprintf(stderr, "p2open failed\n"); else write(1, buf, 16); (void)p2close(fp); }</pre>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

p2open(3GEN)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `fclose(3C)`, `popen(3C)`, `setbuf(3C)`, `attributes(5)`

NOTES Buffered writes on `fp[0]` can make it appear that the command is not listening. Judiciously placed `fflush()` calls or unbuffering `fp[0]` can be a big help; see `fclose(3C)`.

Many commands use buffered output when connected to a pipe. That, too, can make it appear as if things are not working.

Usage is not the same as for `popen()`, although it is closely related.

NAME	pam – PAM (Pluggable Authentication Module)
SYNOPSIS	<pre>#include <security/pam_appl.h> cc [flag...] file ... -lpam [library ...]</pre>
DESCRIPTION	The PAM framework, <code>libpam</code>, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication. PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. This framework also allows new authentication service modules to be plugged in and made available without modifying the applications.
Interface Overview	The PAM library interface consists of six categories of functions, the names for which all start with the prefix <code>pam_</code>. The first category contains functions for establishing and terminating an authentication activity, which are <code>pam_start(3PAM)</code> and <code>pam_end(3PAM)</code>. The functions <code>pam_set_data(3PAM)</code> and <code>pam_get_data(3PAM)</code> maintain module specific data. The functions <code>pam_set_item(3PAM)</code> and <code>pam_get_item(3PAM)</code> maintain state information. <code>pam_strerror(3PAM)</code> is the function that returns error status information. The second category contains the functions that authenticate an individual user and set the credentials of the user, <code>pam_authenticate(3PAM)</code> and <code>pam_setcred(3PAM)</code>. The third category of PAM interfaces is account management. The function <code>pam_acct_mgmt(3PAM)</code> checks for password aging and access-hour restrictions. Category four contains the functions that perform session management after access to the system has been granted. See <code>pam_open_session(3PAM)</code> and <code>pam_close_session(3PAM)</code>. The fifth category consists of the function that changes authentication tokens, <code>pam_chauthtok(3PAM)</code>. An authentication token is the object used to verify the identity of the user. In UNIX, an authentication token is a user's password. The sixth category of functions can be used to set values for PAM environment variables. See <code>pam_putenv(3PAM)</code>, <code>pam_getenv(3PAM)</code>, and <code>pam_getenvlist(3PAM)</code>. The <code>pam_* ()</code> interfaces are implemented through the library <code>libpam</code>. For each of the categories listed above, excluding categories one and six, dynamically loadable shared modules exist that provides the appropriate service layer functionality upon demand. The functional entry points in the service layer start with the <code>pam_sm_</code> prefix. The only difference between the <code>pam_sm_* ()</code> interfaces and their

pam(3PAM)

	corresponding <code>pam_</code> interfaces is that all the <code>pam_sm_*</code>() interfaces require extra parameters to pass service-specific options to the shared modules. Refer to <code>pam_sm(3PAM)</code> for an overview of the PAM service module APIs.
Stateful Interface	A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to <code>pam_start()</code>. <code>pam_start()</code> allocates space, performs various initialization activities, and assigns a PAM authentication handle to be used for subsequent calls to the library. After initiating an authentication transaction, applications can invoke <code>pam_authenticate()</code> to authenticate a particular user, and <code>pam_acct_mgmt()</code> to perform system entry management. For example, the application may want to determine if the user's password has expired. If the user has been successfully authenticated, the application calls <code>pam_setcred()</code> to set any user credentials associated with the authentication service. Within one authentication transaction (between <code>pam_start()</code> and <code>pam_end()</code>), all calls to the PAM interface should be made with the same authentication handle returned by <code>pam_start()</code>. This is necessary because certain service modules may store module-specific data in a handle that is intended for use by other modules. For example, during the call to <code>pam_authenticate()</code>, service modules may store data in the handle that is intended for use by <code>pam_setcred()</code>. To perform session management, applications call <code>pam_open_session()</code>. Specifically, the system may want to store the total time for the session. The function <code>pam_close_session()</code> closes the current session. When necessary, applications can call <code>pam_get_item()</code> and <code>pam_set_item()</code> to access and to update specific authentication information. Such information may include the current username. To terminate an authentication transaction, the application simply calls <code>pam_end()</code>, which frees previously allocated space used to store authentication information.
Application-Authentication Service Interactive Interface	The authentication service in PAM does not communicate directly with the user; instead it relies on the application to perform all such interactions. The application passes a pointer to the function, <code>conv()</code>, along with any associated application data pointers, through a <code>pam_conv</code> structure to the authentication service when it initiates an authentication transaction, via a call to <code>pam_start()</code>. The service will then use the function, <code>conv()</code>, to prompt the user for data, output error messages, and display text information. Refer to <code>pam_start(3PAM)</code> for more information.
Stacking Multiple Schemes	The PAM architecture enables authentication by multiple authentication services through <i>stacking</i>. System entry applications, such as <code>login(1)</code>, stack multiple service modules to authenticate users with multiple authentication services. The order in which authentication service modules are stacked is specified in the configuration file, <code>pam.conf(4)</code>. A system administrator determines this ordering, and also determines whether the same password can be used for all authentication services.

Administrative Interface

The authentication library, `/usr/lib/libpam.so.1`, implements the framework interface. Various authentication services are implemented by their own loadable modules whose paths are specified through the `pam.conf(4)` file.

RETURN VALUES

The PAM functions may return one of the following generic values, or one of the values defined in the specific man pages:

<code>PAM_SUCCESS</code>	The function returned successfully.
<code>PAM_OPEN_ERR</code>	<code>dlopen()</code> failed when dynamically loading a service module.
<code>PAM_SYMBOL_ERR</code>	Symbol not found.
<code>PAM_SERVICE_ERR</code>	Error in service module.
<code>PAM_SYSTEM_ERR</code>	System error.
<code>PAM_BUF_ERR</code>	Memory buffer error.
<code>PAM_CONV_ERR</code>	Conversation failure.
<code>PAM_PERM_DENIED</code>	Permission denied.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

SEE ALSO

`login(1)`, `pam_authenticate(3PAM)`, `pam_chauthtok(3PAM)`, `pam_open_session(3PAM)`, `pam_set_item(3PAM)`, `pam_setcred(3PAM)`, `pam_sm(3PAM)`, `pam_start(3PAM)`, `pam_strerror(3PAM)`, `pam.conf(4)`, `attributes(5)`

NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pam_acct_mgmt(3PAM)

NAME	pam_acct_mgmt – perform PAM account validation procedures								
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_acct_mgmt(pam_handle_t *pamh, int flags);</pre>								
DESCRIPTION	The <code>pam_acct_mgmt()</code> function is called to determine if the current user's account is valid. It checks for password and account expiration, and verifies access hour restrictions. This function is typically called after the user has been authenticated with <code>pam_authenticate(3PAM)</code>. The <i>pamh</i> argument is an authentication handle obtained by a prior call to <code>pam_start()</code>. The following flags may be set in the <i>flags</i> field: <table> <tr> <td>PAM_SILENT</td><td>The account management service should not generate any messages.</td></tr> <tr> <td>PAM_DISALLOW_NULL_AUTHTOK</td><td>The account management service should return PAM_NEW_AUTHTOK_REQD if the user has a null authentication token.</td></tr> </table>	PAM_SILENT	The account management service should not generate any messages.	PAM_DISALLOW_NULL_AUTHTOK	The account management service should return PAM_NEW_AUTHTOK_REQD if the user has a null authentication token.				
PAM_SILENT	The account management service should not generate any messages.								
PAM_DISALLOW_NULL_AUTHTOK	The account management service should return PAM_NEW_AUTHTOK_REQD if the user has a null authentication token.								
RETURN VALUES	Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in <code>pam(3PAM)</code>, the following values may be returned: <table> <tr> <td>PAM_USER_UNKNOWN</td><td>User not known to underlying account management module.</td></tr> <tr> <td>PAM_AUTH_ERR</td><td>Authentication failure.</td></tr> <tr> <td>PAM_NEW_AUTHTOK_REQD</td><td>New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged.</td></tr> <tr> <td>PAM_ACCT_EXPIRED</td><td>User account has expired.</td></tr> </table>	PAM_USER_UNKNOWN	User not known to underlying account management module.	PAM_AUTH_ERR	Authentication failure.	PAM_NEW_AUTHTOK_REQD	New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged.	PAM_ACCT_EXPIRED	User account has expired.
PAM_USER_UNKNOWN	User not known to underlying account management module.								
PAM_AUTH_ERR	Authentication failure.								
PAM_NEW_AUTHTOK_REQD	New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged.								
PAM_ACCT_EXPIRED	User account has expired.								
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Stable								
MT-Level	MT-Safe with exceptions								
SEE ALSO	<code>pam(3PAM)</code> , <code>pam_authenticate(3PAM)</code> , <code>pam_start(3PAM)</code> , <code>libpam(3LIB)</code> , <code>attributes(5)</code>								
NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.								

NAME	pam_authenticate – perform authentication within the PAM framework										
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lpam [<i>library</i> ...] #include <security/pam_appl.h> int pam_authenticate(pam_handle_t *<i>pamh</i>, int <i>flags</i>);</pre>										
DESCRIPTION	The <code>pam_authenticate()</code> function is called to authenticate the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication service configured within the system. The user in question should have been specified by a prior call to <code>pam_start()</code> or <code>pam_set_item()</code>. The following flags may be set in the <i>flags</i> field: <table> <tr> <td>PAM_SILENT</td><td>Authentication service should not generate any messages.</td></tr> <tr> <td>PAM_DISALLOW_NULL_AUTHTOK</td><td>The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.</td></tr> </table>	PAM_SILENT	Authentication service should not generate any messages.	PAM_DISALLOW_NULL_AUTHTOK	The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.						
PAM_SILENT	Authentication service should not generate any messages.										
PAM_DISALLOW_NULL_AUTHTOK	The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.										
RETURN VALUES	Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3PAM), the following values may be returned: <table> <tr> <td>PAM_AUTH_ERR</td><td>Authentication failure.</td></tr> <tr> <td>PAM_CRED_INSUFFICIENT</td><td>Cannot access authentication data due to insufficient credentials.</td></tr> <tr> <td>PAM_AUTHINFO_UNAVAIL</td><td>Underlying authentication service cannot retrieve authentication information.</td></tr> <tr> <td>PAM_USER_UNKNOWN</td><td>User not known to the underlying authentication module.</td></tr> <tr> <td>PAM_MAXTRIES</td><td>An authentication service has maintained a retry count which has been reached. No further retries should be attempted.</td></tr> </table>	PAM_AUTH_ERR	Authentication failure.	PAM_CRED_INSUFFICIENT	Cannot access authentication data due to insufficient credentials.	PAM_AUTHINFO_UNAVAIL	Underlying authentication service cannot retrieve authentication information.	PAM_USER_UNKNOWN	User not known to the underlying authentication module.	PAM_MAXTRIES	An authentication service has maintained a retry count which has been reached. No further retries should be attempted.
PAM_AUTH_ERR	Authentication failure.										
PAM_CRED_INSUFFICIENT	Cannot access authentication data due to insufficient credentials.										
PAM_AUTHINFO_UNAVAIL	Underlying authentication service cannot retrieve authentication information.										
PAM_USER_UNKNOWN	User not known to the underlying authentication module.										
PAM_MAXTRIES	An authentication service has maintained a retry count which has been reached. No further retries should be attempted.										
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Interface Stability	Stable										
MT-Level	MT-Safe with exceptions										
SEE ALSO	pam(3PAM), pam_open_session(3PAM), pam_set_item(3PAM), pam_setcred(3PAM), pam_start(3PAM), libpam(3LIB), attributes(5)										

pam_authenticate(3PAM)

NOTES	In the case of authentication failures due to an incorrect username or password, it is the responsibility of the application to retry <code>pam_authenticate()</code> and to maintain the retry count. An authentication service module may implement an internal retry count and return an error <code>PAM_MAXTRIES</code> if the module does not want the application to retry. If the PAM framework cannot load the authentication module, then it will return <code>PAM_ABORT</code>. This indicates a serious failure, and the application should not attempt to retry the authentication. For security reasons, the location of authentication failures is hidden from the user. Thus, if several authentication services are stacked and a single service fails, <code>pam_authenticate()</code> requires that the user re-authenticate each of the services. A null authentication token in the authentication database will result in successful authentication unless <code>PAM_DISALLOW_NULL_AUTHTOK</code> was specified. In such cases, there will be no prompt to the user to enter an authentication token. The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_chauthtok – perform password related functions within the PAM framework														
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_chauthtok(pam_handle_t *pamh, const int flags);</pre>														
DESCRIPTION	The <code>pam_chauthtok()</code> function is called to change the authentication token associated with a particular user referenced by the authentication handle <i>pamh</i>. The following flag may be passed in to <code>pam_chauthtok()</code>: <table> <tr> <td>PAM_SILENT</td><td>The password service should not generate any messages.</td></tr> <tr> <td>PAM_CHANGE_EXPIRED_AUTHOK</td><td>The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.</td></tr> </table> Upon successful completion of the call, the authentication token of the user will be changed in accordance with the password service configured in the system through <code>pam.conf(4)</code>.	PAM_SILENT	The password service should not generate any messages.	PAM_CHANGE_EXPIRED_AUTHOK	The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.										
PAM_SILENT	The password service should not generate any messages.														
PAM_CHANGE_EXPIRED_AUTHOK	The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.														
RETURN VALUES	Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in <code>pam(3PAM)</code>, the following values may be returned: <table> <tr> <td>PAM_PERM_DENIED</td><td>No permission.</td></tr> <tr> <td>PAM_AUTHOK_ERR</td><td>Authentication token manipulation error.</td></tr> <tr> <td>PAM_AUTHOK_RECOVERY_ERR</td><td>Authentication information cannot be recovered.</td></tr> <tr> <td>PAM_AUTHOK_LOCK_BUSY</td><td>Authentication token lock busy.</td></tr> <tr> <td>PAM_AUTHOK_DISABLE_AGING</td><td>Authentication token aging disabled.</td></tr> <tr> <td>PAM_USER_UNKNOWN</td><td>User unknown to password service.</td></tr> <tr> <td>PAM_TRY_AGAIN</td><td>Preliminary check by password service failed.</td></tr> </table>	PAM_PERM_DENIED	No permission.	PAM_AUTHOK_ERR	Authentication token manipulation error.	PAM_AUTHOK_RECOVERY_ERR	Authentication information cannot be recovered.	PAM_AUTHOK_LOCK_BUSY	Authentication token lock busy.	PAM_AUTHOK_DISABLE_AGING	Authentication token aging disabled.	PAM_USER_UNKNOWN	User unknown to password service.	PAM_TRY_AGAIN	Preliminary check by password service failed.
PAM_PERM_DENIED	No permission.														
PAM_AUTHOK_ERR	Authentication token manipulation error.														
PAM_AUTHOK_RECOVERY_ERR	Authentication information cannot be recovered.														
PAM_AUTHOK_LOCK_BUSY	Authentication token lock busy.														
PAM_AUTHOK_DISABLE_AGING	Authentication token aging disabled.														
PAM_USER_UNKNOWN	User unknown to password service.														
PAM_TRY_AGAIN	Preliminary check by password service failed.														
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions								
ATTRIBUTE TYPE	ATTRIBUTE VALUE														
Interface Stability	Stable														
MT-Level	MT-Safe with exceptions														
SEE ALSO	<code>login(1)</code> , <code>passwd(1)</code> , <code>pam(3PAM)</code> , <code>pam_authenticate(3PAM)</code> , <code>pam_start(3PAM)</code> , <code>attributes</code>														

pam_chauthtok(3PAM)

NOTES | The flag `PAM_CHANGE_EXPIRED_AUTHTOK` is typically used by a login application which has determined that the user's password has aged or expired. Before allowing the user to login, the login application may invoke `pam_chauthtok()` with this flag to allow the user to update the password. Typically, applications such as `passwd(1)` should not use this flag.

The `pam_chauthtok()` functions performs a preliminary check before attempting to update passwords. This check is performed for each password module in the stack as listed in `pam.conf(4)`. The check may include pinging remote name services to determine if they are available. If `pam_chauthtok()` returns `PAM_TRY_AGAIN`, then the check has failed, and passwords are not updated.

The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_getenv – returns the value for a PAM environment name						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lpam [<i>library</i> ...] #include <security/pam_appl.h> char *pam_getenv(pam_handle_t *<i>pamh</i>, const char *<i>name</i>);</pre>						
DESCRIPTION	The <code>pam_getenv()</code> function searches the PAM handle <i>pamh</i> for a value associated with <i>name</i> . If a value is present, <code>pam_getenv()</code> makes a copy of the value and returns a pointer to the copy back to the calling application. If no such entry exists, <code>pam_getenv()</code> returns NULL. It is the responsibility of the calling application to free the memory returned by <code>pam_getenv()</code> .						
RETURN VALUES	If successful, <code>pam_getenv()</code> returns a copy of the <i>value</i> associated with <i>name</i> in the PAM handle; otherwise, it returns a NULL pointer.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>pam(3PAM)</code> , <code>pam_getenvlist(3PAM)</code> , <code>pam_putenv(3PAM)</code> , <code>libpam(3LIB)</code> , <code>attributes(5)</code>						
NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.						

pam_getenvlist(3PAM)

NAME	pam_getenvlist – returns a list of all the PAM environment variables						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> char **pam_getenvlist(pam_handle_t *pamh);</pre>						
DESCRIPTION	The <code>pam_getenvlist()</code> function returns a list of all the PAM environment variables stored in the PAM handle <i>pamh</i> . The list is returned as a null-terminated array of pointers to strings. Each string contains a single PAM environment variable of the form <i>name=value</i> . The list returned is a duplicate copy of all the environment variables stored in <i>pamh</i> . It is the responsibility of the calling application to free the memory returned by <code>pam_getenvlist()</code> .						
RETURN VALUES	If successful, <code>pam_getenvlist()</code> returns in a null-terminated array a copy of all the PAM environment variables stored in <i>pamh</i> . Otherwise, <code>pam_getenvlist()</code> returns a null pointer.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe with exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>pam(3PAM)</code> , <code>pam_getenv(3PAM)</code> , <code>pam_putenv(3PAM)</code> , <code>libpam(3LIB)</code> , <code>attributes(5)</code>						
NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.						

NAME	pam_get_user – PAM routine to retrieve user name						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_get_user(pam_handle_t *pamh, char **user, const char *prompt) ;</pre>						
DESCRIPTION	The <code>pam_get_user()</code> function is used by PAM service modules to retrieve the current user name from the PAM handle. If the user name has not been set with <code>pam_start()</code> or <code>pam_set_item()</code>, the PAM conversation function will be used to prompt the user for the user name with the string "prompt". If <i>prompt</i> is NULL, then <code>pam_get_item()</code> is called and the value of PAM_USER_PROMPT is used for prompting. If the value of PAM_USER_PROMPT is NULL, the following default prompt is used: <pre>Please enter user name:</pre> After the user name is gathered by the conversation function, <code>pam_set_item()</code> is called to set the value of PAM_USER. By convention, applications that need to prompt for a user name should call <code>pam_set_item()</code> and set the value of PAM_USER_PROMPT before calling <code>pam_authenticate()</code>. The service module's <code>pam_sm_authenticate()</code> function will then call <code>pam_get_user()</code> to prompt for the user name. Note that certain PAM service modules, such as a smart card module, may override the value of PAM_USER_PROMPT and pass in their own prompt. Applications that call <code>pam_authenticate()</code> multiple times should set the value of PAM_USER to NULL with <code>pam_set_item()</code> before calling <code>pam_authenticate()</code>, if they want the user to be prompted for a new user name each time. The value of <i>user</i> retrieved by <code>pam_get_user()</code> should not be modified or freed. The item will be released by <code>pam_end()</code>.						
RETURN VALUES	Upon success, <code>pam_get_user()</code> returns PAM_SUCCESS; otherwise it returns an error code. Refer to <code>pam(3PAM)</code> for information on error related return values.						
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>pam(3PAM)</code> , <code>pam_authenticate(3PAM)</code> , <code>pam_end(3PAM)</code> , <code>pam_get_item(3PAM)</code> , <code>pam_set_item(3PAM)</code> , <code>pam_sm(3PAM)</code> , <code>pam_sm_authenticate(3PAM)</code> , <code>pam_start(3PAM)</code> , <code>attributes(5)</code>						

`pam_get_user(3PAM)`

NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pam_open_session(3PAM)

NAME	pam_open_session, pam_close_session – perform PAM session creation and termination operations						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_open_session(pam_handle_t *pamh, int flags); int pam_close_session(pam_handle_t *pamh, int flags);</pre>						
DESCRIPTION	The <code>pam_open_session()</code> function is called after a user has been successfully authenticated. See <code>pam_authenticate(3PAM)</code> and <code>pam_acct_mgmt(3PAM)</code>. It is used to notify the session modules that a new session has been initiated. All programs that use the <code>pam(3PAM)</code> library should invoke <code>pam_open_session()</code> when beginning a new session. Upon termination of this activity, <code>pam_close_session()</code> should be invoked to inform <code>pam(3PAM)</code> that the session has terminated. The <i>pamh</i> argument is an authentication handle obtained by a prior call to <code>pam_start()</code>. The following flag may be set in the <i>flags</i> field for <code>pam_open_session()</code> and <code>pam_close_session()</code>: PAM_SILENT The session service should not generate any messages.						
RETURN VALUES	Upon successful completion, PAM_SUCCESS is returned. In addition to the return values defined in <code>pam(3PAM)</code>, the following value may be returned on error: PAM_SESSION_ERR Cannot make or remove an entry for the specified session.						
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe with exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>getutxent(3C)</code> , <code>pam(3PAM)</code> , <code>pam_acct_mgmt(3PAM)</code> , <code>pam_authenticate(3PAM)</code> , <code>pam_start(3PAM)</code> , <code>attributes(5)</code>						
NOTES	In many instances, the <code>pam_open_session()</code> and <code>pam_close_session()</code> calls may be made by different processes. For example, in UNIX the login process opens a session, while the init process closes the session. In this case, UTMP/WTMP entries may be used to link the call to <code>pam_close_session()</code> with an earlier call to <code>pam_open_session()</code>. This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, <i>pamh</i>. The call to <code>pam_open_session()</code> should precede UTMP/WTMP entry management, and the call to <code>pam_close_session()</code> should follow UTMP/WTMP exit management.						

`pam_open_session(3PAM)`

The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_putenv – change or add a value to the PAM environment																
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_putenv(pam_handle_t *pamh, const char *name_value);</pre>																
DESCRIPTION	The <code>pam_putenv()</code> function sets the value of the PAM environment variable <i>name</i> equal to <i>value</i> either by altering an existing PAM variable or by creating a new one. The <i>name_value</i> argument points to a string of the form <i>name=value</i>. A call to <code>pam_putenv()</code> does not immediately change the environment. All <i>name_value</i> pairs are stored in the PAM handle <i>pamh</i>. An application such as <code>login(1)</code> may make a call to <code>pam_getenv(3PAM)</code> or <code>pam_getenvlist(3PAM)</code> to retrieve the PAM environment variables saved in the PAM handle and set them in the environment if appropriate. <code>login</code> will not set PAM environment values which overwrite the values for <code>SHELL</code>, <code>HOME</code>, <code>LOGNAME</code>, <code>MAIL</code>, <code>CDPATH</code>, <code>IFS</code>, and <code>PATH</code>. Nor will <code>login</code> set PAM environment values which overwrite any value that begins with <code>LD_</code>. If <i>name_value</i> equals <code>NAME=</code>, then the value associated with <code>NAME</code> in the PAM handle will be set to an empty value. If <i>name_value</i> equals <code>NAME</code>, then the environment variable <code>NAME</code> will be removed from the PAM handle.																
RETURN VALUES	The <code>pam_putenv()</code> function may return one of the following values: <table> <tr> <td><code>PAM_SUCCESS</code></td><td>The function returned successfully.</td></tr> <tr> <td><code>PAM_OPEN_ERR</code></td><td><code>dlopen()</code> failed when dynamically loading a service module.</td></tr> <tr> <td><code>PAM_SYMBOL_ERR</code></td><td>Symbol not found.</td></tr> <tr> <td><code>PAM_SERVICE_ERR</code></td><td>Error in service module.</td></tr> <tr> <td><code>PAM_SYSTEM_ERR</code></td><td>System error.</td></tr> <tr> <td><code>PAM_BUF_ERR</code></td><td>Memory buffer error.</td></tr> <tr> <td><code>PAM_CONV_ERR</code></td><td>Conversation failure.</td></tr> <tr> <td><code>PAM_PERM_DENIED</code></td><td>Permission denied.</td></tr> </table>	<code>PAM_SUCCESS</code>	The function returned successfully.	<code>PAM_OPEN_ERR</code>	<code>dlopen()</code> failed when dynamically loading a service module.	<code>PAM_SYMBOL_ERR</code>	Symbol not found.	<code>PAM_SERVICE_ERR</code>	Error in service module.	<code>PAM_SYSTEM_ERR</code>	System error.	<code>PAM_BUF_ERR</code>	Memory buffer error.	<code>PAM_CONV_ERR</code>	Conversation failure.	<code>PAM_PERM_DENIED</code>	Permission denied.
<code>PAM_SUCCESS</code>	The function returned successfully.																
<code>PAM_OPEN_ERR</code>	<code>dlopen()</code> failed when dynamically loading a service module.																
<code>PAM_SYMBOL_ERR</code>	Symbol not found.																
<code>PAM_SERVICE_ERR</code>	Error in service module.																
<code>PAM_SYSTEM_ERR</code>	System error.																
<code>PAM_BUF_ERR</code>	Memory buffer error.																
<code>PAM_CONV_ERR</code>	Conversation failure.																
<code>PAM_PERM_DENIED</code>	Permission denied.																
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions										
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
Interface Stability	Stable																
MT-Level	MT-Safe with exceptions																
SEE ALSO	<code>dlopen(3DL)</code> , <code>pam(3PAM)</code> , <code>pam_getenv(3PAM)</code> , <code>pam_getenvlist(3PAM)</code> , <code>libpam(3LIB)</code> , <code>attributes(5)</code>																

pam_putenv(3PAM)

NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_setcred – modify/delete user credentials for an authentication service										
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_setcred(pam_handle_t *pamh, int flags);</pre>										
DESCRIPTION	The <code>pam_setcred()</code> function is used to establish, modify, or delete user credentials. It is typically called after the user has been authenticated and after a session has been opened. See <code>pam_authenticate(3PAM)</code>, <code>pam_acct_mgmt(3PAM)</code>, and <code>pam_open_session(3PAM)</code>. The user is specified by a prior call to <code>pam_start()</code> or <code>pam_set_item()</code>, and is referenced by the authentication handle, <i>pamh</i>. The following flags may be set in the <i>flags</i> field. Note that the first four flags are mutually exclusive: <table> <tr> <td><code>PAM_ESTABLISH_CRED</code></td><td>Set user credentials for an authentication service.</td></tr> <tr> <td><code>PAM_DELETE_CRED</code></td><td>Delete user credentials associated with an authentication service.</td></tr> <tr> <td><code>PAM_REINITIALIZE_CRED</code></td><td>Reinitialize user credentials.</td></tr> <tr> <td><code>PAM_REFRESH_CRED</code></td><td>Extend lifetime of user credentials.</td></tr> <tr> <td><code>PAM_SILENT</code></td><td>Authentication service should not generate any messages.</td></tr> </table> If no flag is set, <code>PAM_ESTABLISH_CRED</code> is used as the default.	<code>PAM_ESTABLISH_CRED</code>	Set user credentials for an authentication service.	<code>PAM_DELETE_CRED</code>	Delete user credentials associated with an authentication service.	<code>PAM_REINITIALIZE_CRED</code>	Reinitialize user credentials.	<code>PAM_REFRESH_CRED</code>	Extend lifetime of user credentials.	<code>PAM_SILENT</code>	Authentication service should not generate any messages.
<code>PAM_ESTABLISH_CRED</code>	Set user credentials for an authentication service.										
<code>PAM_DELETE_CRED</code>	Delete user credentials associated with an authentication service.										
<code>PAM_REINITIALIZE_CRED</code>	Reinitialize user credentials.										
<code>PAM_REFRESH_CRED</code>	Extend lifetime of user credentials.										
<code>PAM_SILENT</code>	Authentication service should not generate any messages.										
RETURN VALUES	Upon success, <code>pam_setcred()</code> returns <code>PAM_SUCCESS</code>. In addition to the error return values described in <code>pam(3PAM)</code> the following values may be returned upon error: <table> <tr> <td><code>PAM_CRED_UNAVAIL</code></td><td>Underlying authentication service can not retrieve user credentials unavailable.</td></tr> <tr> <td><code>PAM_CRED_EXPIRED</code></td><td>User credentials expired.</td></tr> <tr> <td><code>PAM_USER_UNKNOWN</code></td><td>User unknown to underlying authentication service.</td></tr> <tr> <td><code>PAM_CRED_ERR</code></td><td>Failure setting user credentials.</td></tr> </table>	<code>PAM_CRED_UNAVAIL</code>	Underlying authentication service can not retrieve user credentials unavailable.	<code>PAM_CRED_EXPIRED</code>	User credentials expired.	<code>PAM_USER_UNKNOWN</code>	User unknown to underlying authentication service.	<code>PAM_CRED_ERR</code>	Failure setting user credentials.		
<code>PAM_CRED_UNAVAIL</code>	Underlying authentication service can not retrieve user credentials unavailable.										
<code>PAM_CRED_EXPIRED</code>	User credentials expired.										
<code>PAM_USER_UNKNOWN</code>	User unknown to underlying authentication service.										
<code>PAM_CRED_ERR</code>	Failure setting user credentials.										
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes:										

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

pam_setcred(3PAM)

SEE ALSO pam(3PAM), pam_acct_mgmt(3PAM), pam_authenticate(3PAM),
pam_open_session(3PAM), pam_set_item(3PAM), pam_start(3PAM),
libpam(3LIB), attributes(5)

NOTES The interfaces in libpam are MT-Safe only if each thread within the multithreaded
application uses its own PAM handle.

NAME	pam_set_data, pam_get_data – PAM routines to maintain module specific state						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_set_data(pam_handle_t *pamh, const char *module_data_name, void *data, void (*cleanup) (pam_handle_t *pamh, void *data, int pam_end_status)); int pam_get_data(const pam_handle_t *pamh, const char *module_data_name, const void **data);</pre>						
DESCRIPTION	The <code>pam_set_data()</code> and <code>pam_get_data()</code> functions allow PAM service modules to access and update module specific information as needed. These functions should not be used by applications. The <code>pam_set_data()</code> function stores module specific data within the PAM handle <i>pamh</i>. The <i>module_data_name</i> argument uniquely identifies the data, and the <i>data</i> argument represents the actual data. The <i>module_data_name</i> argument should be unique across all services. The <i>cleanup</i> function frees up any memory used by the <i>data</i> after it is no longer needed, and is invoked by <code>pam_end()</code>. The <i>cleanup</i> function takes as its arguments a pointer to the PAM handle, <i>pamh</i>, a pointer to the actual data, <i>data</i>, and a status code, <i>pam_end_status</i>. The status code determines exactly what state information needs to be purged. If <code>pam_set_data()</code> is called and module data already exists from a prior call to <code>pam_set_data()</code> under the same <i>module_data_name</i>, then the existing <i>data</i> is replaced by the new <i>data</i>, and the existing <i>cleanup</i> function is replaced by the new <i>cleanup</i> function. The <code>pam_get_data()</code> function retrieves module-specific data stored in the PAM handle, <i>pamh</i>, identified by the unique name, <i>module_data_name</i>. The <i>data</i> argument is assigned the address of the requested data. The <i>data</i> retrieved by <code>pam_get_data()</code> should not be modified or freed. The <i>data</i> will be released by <code>pam_end()</code>.						
RETURN VALUES	In addition to the return values listed in <code>pam(3PAM)</code>, the following value may also be returned: PAM_NO_MODULE_DATA No module specific data is present.						
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Stable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>pam(3PAM)</code> , <code>pam_end(3PAM)</code> , <code>libpam(3LIB)</code> , <code>attributes(5)</code>						

pam_set_data(3PAM)

NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_set_item, pam_get_item – authentication information routines for PAM																		
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_set_item(pam_handle_t *pamh, int item_type, const void *item) ; int pam_get_item(const pam_handle_t *pamh, int item_type, void **item) ;</pre>																		
DESCRIPTION	The <code>pam_get_item()</code> and <code>pam_set_item()</code> functions allow applications and PAM service modules to access and to update PAM information as needed. The information is specified by <i>item_type</i>, and can be one of the following: <table> <tr> <td>PAM_SERVICE</td><td>The service name.</td></tr> <tr> <td>PAM_USER</td><td>The user name.</td></tr> <tr> <td>PAM_AUTHTOK</td><td>The user authentication token.</td></tr> <tr> <td>PAM_OLDAUTHTOK</td><td>The old user authentication token.</td></tr> <tr> <td>PAM_TTY</td><td>The tty name.</td></tr> <tr> <td>PAM_RHOST</td><td>The remote host name.</td></tr> <tr> <td>PAM_RUSER</td><td>The remote user name.</td></tr> <tr> <td>PAM_CONV</td><td>The <code>pam_conv</code> structure.</td></tr> <tr> <td>PAM_USER_PROMPT</td><td>The default prompt used by <code>pam_get_user()</code>.</td></tr> </table> For security reasons, the <i>item_type</i> PAM_AUTHTOK and PAM_OLDAUTHTOK are available only to the module providers. The authentication module, account module, and session management module should treat PAM_AUTHTOK as the current authentication token and ignore PAM_OLDAUTHTOK. The password management module should treat PAM_OLDAUTHTOK as the current authentication token and PAM_AUTHTOK as the new authentication token. The <code>pam_set_item()</code> function is passed the authentication handle, <i>pamh</i>, returned by <code>pam_start()</code>, a pointer to the object, <i>item</i>, and its type, <i>item_type</i>. If successful, <code>pam_set_item()</code> copies the item to an internal storage area allocated by the authentication module and returns PAM_SUCCESS. An item that had been previously set will be overwritten by the new value. The <code>pam_get_item()</code> function is passed the authentication handle, <i>pamh</i>, returned by <code>pam_start()</code>, an <i>item_type</i>, and the address of the pointer, <i>item</i>, which is assigned the address of the requested object. The object data is valid until modified by a subsequent call to <code>pam_set_item()</code> for the same <i>item_type</i>, or unless it is modified by any of the underlying service modules. If the item has not been previously set, <code>pam_get_item()</code> returns a null pointer. An <i>item</i> retrieved by <code>pam_get_item()</code> should not be modified or freed. The item will be released by <code>pam_end()</code>.	PAM_SERVICE	The service name.	PAM_USER	The user name.	PAM_AUTHTOK	The user authentication token.	PAM_OLDAUTHTOK	The old user authentication token.	PAM_TTY	The tty name.	PAM_RHOST	The remote host name.	PAM_RUSER	The remote user name.	PAM_CONV	The <code>pam_conv</code> structure.	PAM_USER_PROMPT	The default prompt used by <code>pam_get_user()</code> .
PAM_SERVICE	The service name.																		
PAM_USER	The user name.																		
PAM_AUTHTOK	The user authentication token.																		
PAM_OLDAUTHTOK	The old user authentication token.																		
PAM_TTY	The tty name.																		
PAM_RHOST	The remote host name.																		
PAM_RUSER	The remote user name.																		
PAM_CONV	The <code>pam_conv</code> structure.																		
PAM_USER_PROMPT	The default prompt used by <code>pam_get_user()</code> .																		

pam_set_item(3PAM)

RETURN VALUES Upon success, `pam_get_item()` returns `PAM_SUCCESS`; otherwise it returns an error code. Refer to `pam(3PAM)` for information on error related return values.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

SEE ALSO `pam(3PAM)`, `pam_acct_mgmt(3PAM)`, `pam_authenticate(3PAM)`, `pam_chauthtok(3PAM)`, `pam_get_user(3PAM)`, `pam_open_session(3PAM)`, `pam_setcred(3PAM)`, `pam_start(3PAM)`, `attributes(5)`

NOTES The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_sm – PAM Service Module APIs
SYNOPSIS	<pre>#include <security/pam_appl.h> #include <security/pam_modules.h> cc [flag ...] file ... -lpam [library ...]</pre>
DESCRIPTION	PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. The framework also allows new authentication service modules to be plugged in and made available without modifying the applications. The PAM framework, <code>libpam</code>, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication. This manual page gives an overview of the PAM APIs for the service modules.
Interface Overview	The PAM service module interface consists of functions which can be grouped into four categories. The names for all the authentication library functions start with <code>pam_sm</code>. The only difference between the <code>pam_*()</code> interfaces and their corresponding <code>pam_sm_*()</code> interfaces is that all the <code>pam_sm_*()</code> interfaces require extra parameters to pass service-specific options to the shared modules. They are otherwise identical. The first category contains functions to authenticate an individual user, <code>pam_sm_authenticate(3PAM)</code>, and to set the credentials of the user, <code>pam_sm_setcred(3PAM)</code>. These back-end functions implement the functionality of <code>pam_authenticate(3PAM)</code> and <code>pam_setcred(3PAM)</code> respectively. The second category contains the function to do account management: <code>pam_sm_acct_mgmt(3PAM)</code>. This includes checking for password aging and access-hour restrictions. This back-end function implements the functionality of <code>pam_acct_mgmt(3PAM)</code>. The third category contains the functions <code>pam_sm_open_session(3PAM)</code> and <code>pam_sm_close_session(3PAM)</code> to perform session management after access to the system has been granted. These back-end functions implement the functionality of <code>pam_open_session(3PAM)</code> and <code>pam_close_session(3PAM)</code>, respectively. The fourth category consists a function to change authentication tokens <code>pam_sm_chauthtok(3PAM)</code>. This back-end function implements the functionality of <code>pam_chauthtok(3PAM)</code>.
Stateful Interface	A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to <code>pam_start()</code>. <code>pam_start()</code> allocates space, performs various initialization

pam_sm(3PAM)

activities, and assigns an authentication handle to be used for subsequent calls to the library. Note that the service modules do not get called or initialized when `pam_start()` is called. The modules are loaded and the symbols resolved upon first use of that function.

The PAM handle keeps certain information about the transaction that can be accessed through the `pam_get_item()` API. Though the modules can also use `pam_set_item()` to change any of the item information, it is recommended that nothing be changed except `PAM_AUTHTOK` and `PAM_OLDAUTHTOK`.

If the modules want to store any module specific state information then they can use the `pam_set_data(3PAM)` function to store that information with the PAM handle. The data should be stored with a name which is unique across all modules and module types. For example, `SUNW_PAM_UNIX_AUTH_userid` can be used as a name by the UNIX module to store information about the state of user's authentication. Some modules use this technique to share data across two different module types.

Also, during the call to `pam_authenticate()`, the UNIX module may store the authentication status (success or reason for failure) in the handle, using a unique name such as `SUNW_SECURE_RPC_DATA`. This information is intended for use by `pam_setcred()`.

During the call to `pam_acct_mgmt()`, the account modules may store data in the handle to indicate which passwords have aged. This information is intended for use by `pam_chauthtok()`.

The module can also store a cleanup function associated with the data. The PAM framework calls this cleanup function, when the application calls `pam_end()` to close the transaction.

Interaction with the User

The PAM service modules do not communicate directly with the user; instead they rely on the application to perform all such interactions. The application passes a pointer to the function, `conv()`, along with any associated application data pointers, through the `pam_conv` structure when it initiates an authentication transaction (via a call to `pam_start()`). The service module will then use the function, `conv()`, to prompt the user for data, output error messages, and display text information. Refer to `pam_start(3PAM)` for more information. The modules are responsible for the localization of all messages to the user.

CONVENTIONS

By convention, applications that need to prompt for a user name should call `pam_set_item()` and set the value of `PAM_USER_PROMPT` before calling `pam_authenticate()`. The service module's `pam_sm_authenticate()` function will then call `pam_get_user()` to prompt for the user name. Note that certain PAM service modules (such as a smart card module) may override the value of `PAM_USER_PROMPT` and pass in their own prompt.

Though the PAM framework enforces no rules about the module's names, location, options and such, there are certain conventions that all module providers are expected to follow.

By convention, the modules should be located in the `/usr/lib/security` directory. Additional modules may be located in `/opt/<pkg>/lib`.

By convention, the modules are named `pam_<service_name>_<module_type>.so.1`. If the given module implements more than one module type (for example, `pam_unix.so.1` module), then the `module_type` suffix should be dropped.

For every such module, there should be a corresponding manual page in section 5 which should describe the *module_type* it supports, the functionality of the module, along with the options it supports. The dependencies should be clearly identified to the system administrator. For example, it should be made clear whether this module is a stand-alone module or depends upon the presence of some other module. One should also specify whether this module should come before or after some other module in the stack.

By convention, the modules should support the following options:

<code>debug</code>	Syslog debugging information at <code>LOG_DEBUG</code> level. Be careful as to not log any sensitive information such as passwords.
<code>nowarn</code>	Turn off warning messages such as "password is about to expire."

In addition, it is recommended that the `auth` and the `password` module support the following options:

<code>use_first_pass</code>	Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, return failure and do not prompt the user for a password. Support for this scheme allows the user to type only one password for multiple schemes.
<code>try_first_pass</code>	Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, prompt the user for a password after identifying which type of password (ie. UNIX, etc.) is being requested. Support for this scheme allows the user to try to use only one password for multiple schemes, and type multiple passwords only if necessary.

If an unsupported option is passed to the modules, it should syslog the error at `LOG_ERR` level.

pam_sm(3PAM)

	The permission bits on the service module should be set such that it is not writable by either "group" or "other." The PAM framework will not load the module if the above permission rules are not followed.						
ERROR LOGGING	If there are any errors, the modules should log them using <code>syslog(3C)</code> at the <code>LOG_ERR</code> level.						
RETURN VALUES	The PAM service module functions may return any of the PAM error numbers specified in the specific man pages. It can also return a <code>PAM_IGNORE</code> error number to mean that the PAM framework should ignore this module regardless of whether it is required, optional or sufficient. This error number is normally returned when the module does not want to deal with the given user at all.						
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe with exceptions</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	<code>pam(3PAM)</code>, <code>pam_authenticate(3PAM)</code>, <code>pam_chauthtok(3PAM)</code>, <code>pam_get_user(3PAM)</code>, <code>pam_open_session(3PAM)</code>, <code>pam_setcred(3PAM)</code>, <code>pam_set_item(3PAM)</code>, <code>pam_sm_authenticate(3PAM)</code>, <code>pam_sm_chauthtok(3PAM)</code>, <code>pam_sm_open_session(3PAM)</code>, <code>pam_sm_setcred(3PAM)</code>, <code>pam_start(3PAM)</code>, <code>pam_strerror(3PAM)</code>, <code>syslog(3C)</code>, <code>pam.conf(4)</code>, <code>attributes(5)</code>, <code>pam_authtok_check(5)</code>, <code>pam_authtok_get(5)</code>, <code>pam_authtok_store(5)</code>, <code>pam_dhkeys(5)</code>, <code>pam_passwd_auth(5)</code>, <code>pam_unix(5)</code>, <code>pam_unix_account(5)</code>, <code>pam_unix_auth(5)</code>, <code>pam_unix_session(5)</code>						
NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle. The <code>pam_unix(5)</code> module might not be supported in a future release. Similar functionality is provided by <code>pam_authtok_check(5)</code>, <code>pam_authtok_get(5)</code>, <code>pam_authtok_store(5)</code>, <code>pam_dhkeys(5)</code>, <code>pam_passwd_auth(5)</code>, <code>pam_unix_account(5)</code>, <code>pam_unix_auth(5)</code>, and <code>pam_unix_session(5)</code>.						

NAME	pam_sm_acct_mgmt – service provider implementation for pam_acct_mgmt								
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre>								
DESCRIPTION	In response to a call to pam_acct_mgmt(3PAM), the PAM framework calls pam_sm_acct_mgmt() from the modules listed in the pam.conf(4) file. The account management provider supplies the back-end functionality for this interface function. Applications should not call this API directly. The pam_sm_acct_mgmt() function determines whether or not the current user's account and password are valid. This includes checking for password and account expiration, and valid login times. The user in question is specified by a prior call to pam_start(), and is referenced by the authentication handle, <i>pamh</i>, which is passed as the first argument to pam_sm_acct_mgmt(). The following flags may be set in the <i>flags</i> field: <table> <tr> <td>PAM_SILENT</td><td>The account management service should not generate any messages.</td></tr> <tr> <td>PAM_DISALLOW_NULL_AUTHOK</td><td>The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token.</td></tr> </table> The <i>argc</i> argument represents the number of module options passed in from the configuration file pam.conf(4). <i>argv</i> specifies the module options, which are interpreted and processed by the account management service. Please refer to the specific module man pages for the various available <i>options</i>. If an unknown option is passed to the module, an error should be logged through syslog(3C) and the option ignored. If an account management module determines that the user password has aged or expired, it should save this information as state in the authentication handle, <i>pamh</i>, using pam_set_data(). pam_chauthok() uses this information to determine which passwords have expired.	PAM_SILENT	The account management service should not generate any messages.	PAM_DISALLOW_NULL_AUTHOK	The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token.				
PAM_SILENT	The account management service should not generate any messages.								
PAM_DISALLOW_NULL_AUTHOK	The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token.								
RETURN VALUES	If there are no restrictions to logging in, PAM_SUCCESS is returned. The following error values may also be returned upon error: <table> <tr> <td>PAM_USER_UNKNOWN</td><td>User not known to underlying authentication module.</td></tr> <tr> <td>PAM_NEW_AUTHOK_REQD</td><td>New authentication token required.</td></tr> <tr> <td>PAM_ACCT_EXPIRED</td><td>User account has expired.</td></tr> <tr> <td>PAM_PERM_DENIED</td><td>User denied access to account at this time.</td></tr> </table>	PAM_USER_UNKNOWN	User not known to underlying authentication module.	PAM_NEW_AUTHOK_REQD	New authentication token required.	PAM_ACCT_EXPIRED	User account has expired.	PAM_PERM_DENIED	User denied access to account at this time.
PAM_USER_UNKNOWN	User not known to underlying authentication module.								
PAM_NEW_AUTHOK_REQD	New authentication token required.								
PAM_ACCT_EXPIRED	User account has expired.								
PAM_PERM_DENIED	User denied access to account at this time.								

pam_sm_acct_mgmt(3PAM)

PAM_IGNORE

Ignore underlying account module regardless of whether the control flag is *required*, *optional* or *sufficient*.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

SEE ALSO

pam(3PAM), pam_acct_mgmt(3PAM), pam_set_data(3PAM), pam_start(3PAM), syslog(3C), libpam(3LIB), pam.conf(4), attributes(5)

NOTES

The interfaces in libpam are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_sm_authenticate – service provider implementation for pam_authenticate						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre>						
DESCRIPTION	In response to a call to <code>pam_authenticate(3PAM)</code>, the PAM framework calls <code>pam_sm_authenticate()</code> from the modules listed in the <code>pam.conf(4)</code> file. The authentication provider supplies the back-end functionality for this interface function. The <code>pam_sm_authenticate()</code> function is called to verify the identity of the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication scheme configured within the system. The user in question is specified by a prior call to <code>pam_start()</code>, and is referenced by the authentication handle <i>pamh</i>. If the user is unknown to the authentication service, the service module should mask this error and continue to prompt the user for a password. It should then return the error, <code>PAM_USER_UNKNOWN</code>. The following flag may be passed in to <code>pam_sm_authenticate()</code>: <table> <tr> <td><code>PAM_SILENT</code></td><td>The authentication service should not generate any messages.</td></tr> <tr> <td><code>PAM_DISALLOW_NULL_AUTHTOK</code></td><td>The authentication service should return</td></tr> <tr> <td><code>PAM_AUTH_ERROR</code></td><td>The user has a null authentication token.</td></tr> </table> The <i>argc</i> argument represents the number of module options passed in from the configuration file <code>pam.conf(4)</code>. <i>argv</i> specifies the module options, which are interpreted and processed by the authentication service. Please refer to the specific module man pages for the various available <i>options</i>. If any unknown option is passed in, the module should log the error and ignore the option. Before returning, <code>pam_sm_authenticate()</code> should call <code>pam_get_item()</code> and retrieve <code>PAM_AUTHTOK</code>. If it has not been set before and the value is <code>NULL</code>, <code>pam_sm_authenticate()</code> should set it to the password entered by the user using <code>pam_set_item()</code>. An authentication module may save the authentication status (success or reason for failure) as state in the authentication handle using <code>pam_set_data(3PAM)</code>. This information is intended for use by <code>pam_setcred()</code>.	<code>PAM_SILENT</code>	The authentication service should not generate any messages.	<code>PAM_DISALLOW_NULL_AUTHTOK</code>	The authentication service should return	<code>PAM_AUTH_ERROR</code>	The user has a null authentication token.
<code>PAM_SILENT</code>	The authentication service should not generate any messages.						
<code>PAM_DISALLOW_NULL_AUTHTOK</code>	The authentication service should return						
<code>PAM_AUTH_ERROR</code>	The user has a null authentication token.						
RETURN VALUES	Upon successful completion, <code>PAM_SUCCESS</code> must be returned. In addition, the following values may be returned: <table> <tr> <td><code>PAM_MAXTRIES</code></td><td>Maximum number of authentication attempts exceeded.</td></tr> </table>	<code>PAM_MAXTRIES</code>	Maximum number of authentication attempts exceeded.				
<code>PAM_MAXTRIES</code>	Maximum number of authentication attempts exceeded.						

pam_sm_authenticate(3PAM)

PAM_AUTH_ERR	Authentication failure.
PAM_CRED_INSUFFICIENT	Cannot access authentication data due to insufficient credentials.
PAM_AUTHINFO_UNAVAIL	Underlying authentication service can not retrieve authentication information.
PAM_USER_UNKNOWN	User not known to underlying authentication module.
PAM_IGNORE	Ignore underlying authentication module regardless of whether the control flag is <i>required</i> , <i>optional</i> , or <i>sufficient</i> 1.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

SEE ALSO `pam(3PAM)`, `pam_authenticate(3PAM)`, `pam_get_item(3PAM)`, `pam_set_data(3PAM)`, `pam_set_item(3PAM)`, `pam_setcred(3PAM)`, `pam_start(3PAM)`, `libpam(3LIB)`, `pam.conf(4)`, `attributes(5)`

NOTES Modules should not retry the authentication in the event of a failure. Applications handle authentication retries and maintain the retry count. To limit the number of retries, the module can return a `PAM_MAXTRIES` error.

The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME	pam_sm_chauthtok – service provider implementation for pam_chauthtok								
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre>								
DESCRIPTION	In response to a call to <code>pam_chauthtok()</code> the PAM framework calls <code>pam_sm_chauthtok(3PAM)</code> from the modules listed in the <code>pam.conf(4)</code> file. The password management provider supplies the back-end functionality for this interface function. The <code>pam_sm_chauthtok()</code> function changes the authentication token associated with a particular user referenced by the authentication handle <i>pamh</i>. The following flag may be passed to <code>pam_chauthtok()</code>: <table> <tr> <td>PAM_SILENT</td><td>The password service should not generate any messages.</td></tr> <tr> <td>PAM_CHANGE_EXPIRED_AUTH Tok</td><td>The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.</td></tr> <tr> <td>PAM_PRELIM_CHECK</td><td>The password service should only perform preliminary checks. No passwords should be updated.</td></tr> <tr> <td>PAM_UPDATE_AUTH Tok</td><td>The password service should update passwords.</td></tr> </table> Note that PAM_PRELIM_CHECK and PAM_UPDATE_AUTH Tok cannot be set at the same time. Upon successful completion of the call, the authentication token of the user will be ready for change or will be changed, depending upon the flag, in accordance with the authentication scheme configured within the system. The <i>argc</i> argument represents the number of module options passed in from the configuration file <code>pam.conf(4)</code>. The <i>argv</i> argument specifies the module options, which are interpreted and processed by the password management service. Please refer to the specific module man pages for the various available <i>options</i>. It is the responsibility of <code>pam_sm_chauthtok()</code> to determine if the new password meets certain strength requirements. <code>pam_sm_chauthtok()</code> may continue to re-prompt the user (for a limited number of times) for a new password until the password entered meets the strength requirements.	PAM_SILENT	The password service should not generate any messages.	PAM_CHANGE_EXPIRED_AUTH Tok	The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.	PAM_PRELIM_CHECK	The password service should only perform preliminary checks. No passwords should be updated.	PAM_UPDATE_AUTH Tok	The password service should update passwords.
PAM_SILENT	The password service should not generate any messages.								
PAM_CHANGE_EXPIRED_AUTH Tok	The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.								
PAM_PRELIM_CHECK	The password service should only perform preliminary checks. No passwords should be updated.								
PAM_UPDATE_AUTH Tok	The password service should update passwords.								

pam_sm_chauthtok(3PAM)

	Before returning, <code>pam_sm_chauthtok()</code> should call <code>pam_get_item()</code> and retrieve both <code>PAM_AUTHTOK</code> and <code>PAM_OLDAUTHTOK</code> . If both are <code>NULL</code> , <code>pam_sm_chauthtok()</code> should set them to the new and old passwords as entered by the user.														
RETURN VALUES	Upon successful completion, <code>PAM_SUCCESS</code> must be returned. The following values may also be returned: <table><tr><td><code>PAM_PERM_DENIED</code></td><td>No permission.</td></tr><tr><td><code>PAM_AUTHTOK_ERR</code></td><td>Authentication token manipulation error.</td></tr><tr><td><code>PAM_AUTHTOK_RECOVERY_ERR</code></td><td>Old authentication token cannot be recovered.</td></tr><tr><td><code>PAM_AUTHTOK_LOCK_BUSY</code></td><td>Authentication token lock busy.</td></tr><tr><td><code>PAM_AUTHTOK_DISABLE_AGING</code></td><td>Authentication token aging disabled.</td></tr><tr><td><code>PAM_USER_UNKNOWN</code></td><td>User unknown to password service.</td></tr><tr><td><code>PAM_TRY_AGAIN</code></td><td>Preliminary check by password service failed.</td></tr></table>	<code>PAM_PERM_DENIED</code>	No permission.	<code>PAM_AUTHTOK_ERR</code>	Authentication token manipulation error.	<code>PAM_AUTHTOK_RECOVERY_ERR</code>	Old authentication token cannot be recovered.	<code>PAM_AUTHTOK_LOCK_BUSY</code>	Authentication token lock busy.	<code>PAM_AUTHTOK_DISABLE_AGING</code>	Authentication token aging disabled.	<code>PAM_USER_UNKNOWN</code>	User unknown to password service.	<code>PAM_TRY_AGAIN</code>	Preliminary check by password service failed.
<code>PAM_PERM_DENIED</code>	No permission.														
<code>PAM_AUTHTOK_ERR</code>	Authentication token manipulation error.														
<code>PAM_AUTHTOK_RECOVERY_ERR</code>	Old authentication token cannot be recovered.														
<code>PAM_AUTHTOK_LOCK_BUSY</code>	Authentication token lock busy.														
<code>PAM_AUTHTOK_DISABLE_AGING</code>	Authentication token aging disabled.														
<code>PAM_USER_UNKNOWN</code>	User unknown to password service.														
<code>PAM_TRY_AGAIN</code>	Preliminary check by password service failed.														
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe with exceptions</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions								
ATTRIBUTE TYPE	ATTRIBUTE VALUE														
Interface Stability	Stable														
MT-Level	MT-Safe with exceptions														
SEE ALSO	<code>ping(1M)</code> , <code>pam(3PAM)</code> , <code>pam_chauthtok(3PAM)</code> , <code>pam_get_data(3PAM)</code> , <code>pam_get_item(3PAM)</code> , <code>pam_set_data(3PAM)</code> , <code>libpam(3LIB)</code> , <code>pam.conf(4)</code> , <code>attributes(5)</code>														
NOTES	The PAM framework invokes the password services twice. The first time the modules are invoked with the flag, <code>PAM_PRELIM_CHECK</code>. During this stage, the password modules should only perform preliminary checks. For example, they may ping remote name services to see if they are ready for updates. If a password module detects a transient error such as a remote name service temporarily down, it should return <code>PAM_TRY_AGAIN</code> to the PAM framework, which will immediately return the error back to the application. If all password modules pass the preliminary check, the PAM framework invokes the password services again with the flag, <code>PAM_UPDATE_AUTHTOK</code>. During this stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.														

`pam_sm_chauthtok(3PAM)`

If a service module receives the flag `PAM_CHANGE_EXPIRED_AUTH Tok`, it should check whether the password has aged or expired. If the password has aged or expired, then the service module should proceed to update the password. If the status indicates that the password has not yet aged or expired, then the password module should return `PAM_IGNORE`.

If a user's password has aged or expired, a PAM account module could save this information as state in the authentication handle, *pamh*, using `pam_set_data()`. The related password management module could retrieve this information using `pam_get_data()` to determine whether or not it should prompt the user to update the password for this particular module.

The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pam_sm_open_session(3PAM)

NAME	pam_sm_open_session, pam_sm_close_session – service provider implementation for pam_open_session and pam_close_session						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv); int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre>						
DESCRIPTION	In response to a call to pam_open_session(3PAM) and pam_close_session(3PAM), the PAM framework calls pam_sm_open_session() and pam_sm_close_session(), respectively from the modules listed in the pam.conf(4) file. The session management provider supplies the back-end functionality for this interface function. The pam_sm_open_session() function is called to initiate session management. The pam_sm_close_session() function is invoked when a session has terminated. The argument <i>pamh</i> is an authentication handle. The following flag may be set in the <i>flags</i> field: PAM_SILENT Session service should not generate any messages. The <i>argc</i> argument represents the number of module options passed in from the configuration file pam.conf(4). <i>argv</i> specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged through syslog(3C) and the option ignored.						
RETURN VALUES	Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error: <table><tr><td>PAM_SESSION_ERR</td><td>Cannot make or remove an entry for the specified session.</td></tr><tr><td>PAM_IGNORE</td><td>Ignore underlying session module regardless of whether the control flag is <i>required</i>, <i>optional</i> or <i>sufficient</i>.</td></tr></table>	PAM_SESSION_ERR	Cannot make or remove an entry for the specified session.	PAM_IGNORE	Ignore underlying session module regardless of whether the control flag is <i>required</i> , <i>optional</i> or <i>sufficient</i> .		
PAM_SESSION_ERR	Cannot make or remove an entry for the specified session.						
PAM_IGNORE	Ignore underlying session module regardless of whether the control flag is <i>required</i> , <i>optional</i> or <i>sufficient</i> .						
ATTRIBUTES	See attributes(5) for description of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Stable</td></tr><tr><td>MT-Level</td><td>MT-Safe with exceptions</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Stable	MT-Level	MT-Safe with exceptions
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Stable						
MT-Level	MT-Safe with exceptions						
SEE ALSO	pam(3PAM), pam_open_session(3PAM), syslog(3C), libpam(3LIB), pam.conf(4), attributes(5)						

`pam_sm_open_session(3PAM)`

NOTES The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pam_sm_setcred(3PAM)

NAME	pam_sm_setcred – service provider implementation for pam_setcred										
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre>										
DESCRIPTION	In response to a call to <code>pam_setcred(3PAM)</code>, the PAM framework calls <code>pam_sm_setcred()</code> from the modules listed in the <code>pam.conf(4)</code> file. The authentication provider supplies the back-end functionality for this interface function. The <code>pam_sm_setcred()</code> function is called to set the credentials of the current user associated with the authentication handle, <i>pamh</i>. The following flags may be set in the <i>flags</i> field. Note that the first four flags are mutually exclusive: <table><tr><td>PAM_ESTABLISH_CRED</td><td>Set user credentials for the authentication service.</td></tr><tr><td>PAM_DELETE_CRED</td><td>Delete user credentials associated with the authentication service.</td></tr><tr><td>PAM_REINITIALIZE_CRED</td><td>Reinitialize user credentials.</td></tr><tr><td>PAM_REFRESH_CRED</td><td>Extend lifetime of user credentials.</td></tr><tr><td>PAM_SILENT</td><td>Authentication service should not generate messages</td></tr></table> If no flag is set, <code>PAM_ESTABLISH_CRED</code> is used as the default. The <i>argc</i> argument represents the number of module options passed in from the configuration file <code>pam.conf(4)</code>. <i>argv</i> specifies the module options, which are interpreted and processed by the authentication service. If an unknown option is passed to the module, an error should be logged and the option ignored. If the <code>PAM_SILENT</code> flag is not set, then <code>pam_sm_setcred()</code> should print any failure status from the corresponding <code>pam_sm_authenticate()</code> function using the conversation function. The authentication status (success or reason for failure) is saved as module-specific state in the authentication handle by the authentication module. The status should be retrieved using <code>pam_get_data()</code>, and used to determine if user credentials should be set.	PAM_ESTABLISH_CRED	Set user credentials for the authentication service.	PAM_DELETE_CRED	Delete user credentials associated with the authentication service.	PAM_REINITIALIZE_CRED	Reinitialize user credentials.	PAM_REFRESH_CRED	Extend lifetime of user credentials.	PAM_SILENT	Authentication service should not generate messages
PAM_ESTABLISH_CRED	Set user credentials for the authentication service.										
PAM_DELETE_CRED	Delete user credentials associated with the authentication service.										
PAM_REINITIALIZE_CRED	Reinitialize user credentials.										
PAM_REFRESH_CRED	Extend lifetime of user credentials.										
PAM_SILENT	Authentication service should not generate messages										
RETURN VALUES	Upon successful completion, <code>PAM_SUCCESS</code> should be returned. The following values may also be returned upon error: <table><tr><td>PAM_CRED_UNAVAIL</td><td>Underlying authentication service can not retrieve user credentials.</td></tr><tr><td>PAM_CRED_EXPIRED</td><td>User credentials have expired.</td></tr></table>	PAM_CRED_UNAVAIL	Underlying authentication service can not retrieve user credentials.	PAM_CRED_EXPIRED	User credentials have expired.						
PAM_CRED_UNAVAIL	Underlying authentication service can not retrieve user credentials.										
PAM_CRED_EXPIRED	User credentials have expired.										

pam_sm_setcred(3PAM)

PAM_USER_UNKNOWN	User unknown to the authentication service.
PAM_CRED_ERR	Failure in setting user credentials.
PAM_IGNORE	Ignore underlying authentication module regardless of whether the control flag is <i>required</i> , <i>optional</i> , or <i>sufficient</i> .

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

SEE ALSO pam(3PAM), pam_authenticate(3PAM), pam_get_data(3PAM), pam_setcred(3PAM), pam_sm_authenticate(3PAM), libpam(3LIB), pam.conf(4), attributes(5)

NOTES The pam_sm_setcred() function is passed the same module options that are used by pam_sm_authenticate().

The interfaces in libpam are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pam_start(3PAM)

NAME	pam_start, pam_end – authentication transaction routines for PAM						
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_start(const char *service, const char *user, const struct pam_conv *pam_conv, pam_handle_t **pamh); int pam_end(pam_handle_t *pamh, int status);</pre>						
DESCRIPTION	The <code>pam_start()</code> function is called to initiate an authentication transaction. <code>pam_start()</code> takes as arguments the name of the current service, <i>service</i>, the name of the user to be authenticated, <i>user</i>, the address of the conversation structure, <i>pam_conv</i>, and the address of a variable to be assigned the authentication handle <i>pamh</i>. Upon successful completion, <i>pamh</i> refers to a PAM handle for use with subsequent calls to the authentication library. The <i>pam_conv</i> structure contains the address of the conversation function provided by the application. The underlying PAM service module invokes this function to output information to and retrieve input from the user. The <i>pam_conv</i> structure has the following entries: <pre>struct pam_conv { int (*conv)(); /* Conversation function */ void *appdata_ptr; /* Application data */ }; int conv(int num_msg, const struct pam_message **msg, struct pam_response **resp, void *appdata_ptr);</pre> The <code>conv()</code> function is called by a service module to hold a PAM conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction. The <i>num_msg</i> parameter is the number of messages associated with the call. The parameter <i>msg</i> is a pointer to an array of length <i>num_msg</i> of the <i>pam_message</i> structure. The <i>pam_message</i> structure is used to pass prompt, error message, or any text information from the authentication service to the application or user. It is the responsibility of the PAM service modules to localize the messages. The memory used by <i>pam_message</i> has to be allocated and freed by the PAM modules. The <i>pam_message</i> structure has the following entries: <pre>struct pam_message{ int msg_style; char *msg; };</pre> The message style, <i>msg_style</i>, can be set to one of the following values: <table><tr><td>PAM_PROMPT_ECHO_OFF</td><td>Prompt user, disabling echoing of response.</td></tr><tr><td>PAM_PROMPT_ECHO_ON</td><td>Prompt user, enabling echoing of response.</td></tr><tr><td>PAM_ERROR_MSG</td><td>Print error message.</td></tr></table>	PAM_PROMPT_ECHO_OFF	Prompt user, disabling echoing of response.	PAM_PROMPT_ECHO_ON	Prompt user, enabling echoing of response.	PAM_ERROR_MSG	Print error message.
PAM_PROMPT_ECHO_OFF	Prompt user, disabling echoing of response.						
PAM_PROMPT_ECHO_ON	Prompt user, enabling echoing of response.						
PAM_ERROR_MSG	Print error message.						

pam_start(3PAM)

PAM_TEXT_INFO	Print general text information.
PAM_MSG_NOCONF	Print general text information without user acknowledgment.
PAM_CONV_INTERRUPT	Return from the conversation function.

The maximum size of the message and the response string is PAM_MAX_MSG_SIZE as defined in <security/pam.appl.h>.

The structure *pam_response* is used by the authentication service to get the user's response back from the application or user. The storage used by *pam_response* has to be allocated by the application and freed by the PAM modules. The *pam_response* structure has the following entries:

```
struct pam_response{
    char *resp;
    int  resp_retcode; /* currently not used, */
                          /* should be set to 0 */
};
```

It is the responsibility of the conversation function to strip off NEWLINE characters for PAM_PROMPT_ECHO_OFF and PAM_PROMPT_ECHO_ON message styles, and to add NEWLINE characters (if appropriate) for PAM_ERROR_MSG and PAM_TEXT_INFO message styles.

The *appdata_ptr* argument is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

The *pam_end()* function is called to terminate the authentication transaction identified by *pamh* and to free any storage area allocated by the authentication module. The argument, *status*, is passed to the *cleanup()* function stored within the *pam* handle, and is used to determine what module-specific state must be purged. A cleanup function is attached to the handle by the underlying PAM modules through a call to *pam_set_data(3PAM)* to free module-specific data.

RETURN VALUES

Refer to *pam(3PAM)* for information on error related return values.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

pam_start(3PAM)

SEE ALSO | libpam(3LIB), pam(3PAM), pam_acct_mgmt(3PAM), pam_authenticate(3PAM),
pam_chauthtok(3PAM), pam_open_session(3PAM), pam_setcred(3PAM),
pam_set_data(3PAM), pam_strerror(3PAM), attributes(5)

NOTES | The interfaces in libpam are MT-Safe only if each thread within the multithreaded
application uses its own PAM handle.

NAME	pam_strerror – get PAM error message string
SYNOPSIS	<pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> const char *pam_strerror(pam_handle_t *pamh, int errnum);</pre>
DESCRIPTION	The <code>pam_strerror()</code> function maps the PAM error number in <i>errnum</i> to a PAM error message string, and returns a pointer to that string. The application should not free or modify the string returned. The <i>pamh</i> argument is the PAM handle obtained by a prior call to <code>pam_start()</code>. If <code>pam_start()</code> returns an error, a null PAM handle should be passed.
ERRORS	The <code>pam_strerror()</code> function returns NULL if <i>errnum</i> is out-of-range.
ATTRIBUTES	See <code>attributes(5)</code> for description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable
MT-Level	MT-Safe with exceptions

SEE ALSO	<code>pam(3PAM)</code> , <code>pam_start(3PAM)</code> , <code>attributes(5)</code>
NOTES	The interfaces in <code>libpam</code> are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

pathfind(3GEN)

NAME	pathfind – search for named file in named directories																										
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *pathfind(const char *<i>path</i>, const char *<i>name</i>, const char *<i>mode</i>) ;</pre>																										
DESCRIPTION	The <code>pathfind()</code> function searches the directories named in <i>path</i> for the file <i>name</i>. The directories named in <i>path</i> are separated by colons (:). The <i>mode</i> argument is a string of option letters chosen from the set <code>[rwxfbcdpugks]</code> : <table><thead><tr><th>Letter</th><th>Meaning</th></tr></thead><tbody><tr><td>r</td><td>readable</td></tr><tr><td>w</td><td>writable</td></tr><tr><td>x</td><td>executable</td></tr><tr><td>f</td><td>normal file</td></tr><tr><td>b</td><td>block special</td></tr><tr><td>c</td><td>character special</td></tr><tr><td>d</td><td>directory</td></tr><tr><td>p</td><td>FIFO (pipe)</td></tr><tr><td>u</td><td>set user ID bit</td></tr><tr><td>g</td><td>set group ID bit</td></tr><tr><td>k</td><td>sticky bit</td></tr><tr><td>s</td><td>size non-zero</td></tr></tbody></table> Options read, write, and execute are checked relative to the real (not the effective) user ID and group ID of the current process. If <i>name</i> begins with a slash, it is treated as an absolute path name, and <i>path</i> is ignored. An empty <i>path</i> member is treated as the current directory. A slash (/) character is not prepended at the occurrence of the first match; rather, the unadorned <i>name</i> is returned.	Letter	Meaning	r	readable	w	writable	x	executable	f	normal file	b	block special	c	character special	d	directory	p	FIFO (pipe)	u	set user ID bit	g	set group ID bit	k	sticky bit	s	size non-zero
Letter	Meaning																										
r	readable																										
w	writable																										
x	executable																										
f	normal file																										
b	block special																										
c	character special																										
d	directory																										
p	FIFO (pipe)																										
u	set user ID bit																										
g	set group ID bit																										
k	sticky bit																										
s	size non-zero																										
EXAMPLES	EXAMPLE 1 Example of finding the <code>ls</code> command using the <code>PATH</code> environment variable. To find the <code>ls</code> command using the <code>PATH</code> environment variable: <pre>pathfind (getenv ("PATH"), "ls", "rx")</pre>																										
RETURN VALUES	The <code>pathfind()</code> function returns a <code>(char *)</code> value containing static, thread-specific data that will be overwritten upon the next call from the same thread.																										

pathfind(3GEN)

If the file *name* with all characteristics specified by *mode* is found in any of the directories specified by *path*, then `pathfind()` returns a pointer to a string containing the member of *path*, followed by a slash character (/), followed by *name*.

If no match is found, `pathname()` returns a null pointer, `((char *) 0)`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `sh(1)`, `test(1)`, `access(2)`, `mknod(2)`, `stat(2)`, `getenv(3C)`, `attributes(5)`

NOTES The string pointed to by the returned pointer is stored in an area that is reused on subsequent calls to `pathfind()`. The string should not be deallocated by the caller.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

pctx_capture(3CPC)

NAME	pctx_capture, pctx_create, pctx_run, pctx_release – process context library
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpctx [<i>library...</i>] #include <libpctx.h> typedef void (pctx_errfn_t) (const char *<i>fn</i>, const char *<i>fnt</i>, va_list <i>ap</i>); pctx_t *pctx_create(const char *<i>filename</i>, char *const *<i>argv</i>, void *<i>arg</i>, int <i>verbose</i>, pctx_errfn_t *<i>errfn</i>); pctx_t *pctx_capture(pid_t <i>pid</i>, void *<i>arg</i>, int <i>verbose</i>, pctx_errfn_t *<i>errfn</i>); int pctx_run(pctx_t *<i>pctx</i>, uint_t <i>sample</i>, uint_t <i>nsamples</i>, int (*<i>tick</i>) (pctx *, pid_t, id_t, void *)); void pctx_release(pctx_t *<i>pctx</i>);</pre>
DESCRIPTION	This family of functions allows a controlling process (the process that invokes them) to create or capture controlled processes. The functions allow the occurrence of various events of interest in the controlled process to cause the controlled process to be stopped, and to cause callback routines to be invoked in the controlling process.
pctx_create() and pctx_capture()	There are two ways a process can be acquired by the process context functions. First, a named application can be invoked with the usual <i>argv</i>[] array using pctx_create(), which forks the caller and execs the application in the child. Alternatively, an existing process can be captured by its process ID using pctx_capture(). Both functions accept a pointer to an opaque handle, <i>arg</i>; this is saved and treated as a caller-private handle that is passed to the other functions in the library. Both functions accept a pointer to a fork(3C)-like error routine <i>errfn</i>; a default version is provided if NULL is specified. A freshly-created process is created stopped; similarly, a process that has been successfully captured is stopped by the act of capturing it, thereby allowing the caller to specify the handlers that should be called when various events occur in the controlled process. The set of handlers is listed on the pctx_set_events(3CPC) manual page.
pctx_run()	Once the callback handlers have been set with pctx_set_events(), the application can be set running using pctx_run(). This function starts the event handling loop; it returns only when either the process has exited, the number of time samples has expired, or an error has occurred (for example, if the controlling process is not privileged, and the controlled process has exec-ed a setuid program). Every <i>sample</i> milliseconds the process is stopped and the <i>tick()</i> routine is called so that, for example, the performance counters can be sampled by the caller. No periodic sampling is performed if <i>sample</i> is 0.

pctx_capture(3CPC)

pctx_release() Once pctx_run() has returned, the process can be released and the underlying storage freed using pctx_release(). Releasing the process will either allow the controlled process to continue (in the case of an existing captured process and its children) or kill the process (if it and its children were created using pctx_create()).

RETURN VALUES Upon successful completion, pctx_capture() and pctx_create() return a valid handle. Otherwise, the functions print a diagnostic message and return NULL.

Upon successful completion, pctx_run() returns 0 with the controlled process either stopped or exited (if the controlled process has invoked exit(2).) If an error has occurred (for example, if the controlled process has exec-ed a set-ID executable, if certain callbacks have returned error indications, or if the process was unable to respond to proc(4) requests) an error message is printed and the function returns -1.

USAGE Within an event handler in the controlling process, the controlled process can be made to perform various system calls on its behalf. No system calls are directly supported in this version of the API, though system calls are executed by the cpc_pctx family of interfaces in libcpc such as cpc_pctx_bind_event(3CPC). A specially created agent LWP is used to execute these system calls in the controlled process. See proc(4) for more details.

While executing the event handler functions, the library arranges for the signals SIGTERM, SIGQUIT, SIGABRT, and SIGINT to be blocked to reduce the likelihood of a keyboard signal killing the controlling process prematurely, thereby leaving the controlled process permanently stopped while the agent LWP is still alive inside the controlled process.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO fork(2), cpc(3CPC), pctx_set_events(3CPC), proc(4), attributes(5).

pctx_set_events(3CPC)

NAME	pctx_set_events – associate callbacks with process events
SYNOPSIS	<pre>cc [flag...] file... -lpctx [library...] #include <libpctx.h> typedef enum { PCTX_NULL_EVENT = 0, PCTX_SYSC_EXEC_EVENT, PCTX_SYSC_FORK_EVENT, PCTX_SYSC_EXIT_EVENT, PCTX_SYSC_LWP_CREATE_EVENT, PCTX_INIT_LWP_EVENT, PCTX_FINI_LWP_EVENT, PCTX_SYSC_LWP_EXIT_EVENT } pctx_event_t; typedef int pctx_sysc_execfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, char *cmd, void *arg); typedef void pctx_sysc_forkfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, pid_t child, void *arg); typedef void pctx_sysc_exitfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, void *arg); typedef int pctx_sysc_lwp_createfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, void *arg); typedef int pctx_init_lwpfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, void *arg); typedef int pctx_fini_lwpfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, void *arg); typedef int pctx_sysc_lwp_exitfn_t(pctx_t *pctx, pid_t pid, id_t lwpid, void *arg); int pctx_set_events(pctx_t *pctx, ...);</pre>
DESCRIPTION	The <code>pctx_set_events()</code> function allows the caller (the controlling process) to express interest in various events in the controlled process. See <code>pctx_capture(3CPC)</code> for information about how the controlling process is able to create, capture and manipulate the controlled process. The <code>pctx_set_events()</code> function takes a <code>pctx_t</code> handle, followed by a variable length list of pairs of <code>pctx_event_t</code> tags and their corresponding handlers, terminated by a <code>PCTX_NULL_EVENT</code> tag. Most of the events correspond closely to various classes of system calls, though two additional pseudo-events (<i>init_lwp</i> and <i>fini_lwp</i>) are provided to allow callers to perform various housekeeping tasks. The <i>init_lwp</i> handler is called as soon as the library identifies a new LWP, while <i>fini_lwp</i> is called just before the LWP disappears.

Thus the classic "hello world" program would see an *init_lwp* event, a *fini_lwp* event and (process) *exit* event, in that order. The table below displays the interactions between the states of the controlled process and the handlers executed by users of the library.

System Calls and pctx Handlers		
System call	Handler	Comments
exec, execve	<i>fini_lwp</i>	Invoked serially on all lwps in the process.
	<i>exec</i>	Only invoked if the <code>exec()</code> system call succeeded.
	<i>init_lwp</i>	If the <code>exec</code> succeeds, only invoked on lwp 1. If the <code>exec</code> fails, invoked serially on all lwps in the process.
fork, vfork, fork1	<i>fork</i>	Only invoked if the <code>fork()</code> system call succeeded.
exit	<i>fini_lwp</i>	Invoked on all lwps in the process.
	<i>exit</i>	Invoked on the exiting lwp.
_lwp_create	<i>init_lwp</i>	Only if the corresponding <code>_lwp_create()</code> system call succeeded.
	<i>lwp_create</i>	
_lwp_exit	<i>fini_lwp</i>	
	<i>lwp_exit</i>	

Each of the handlers is passed the caller's opaque handle, a `pctx_t` handle, the `pid`, and `lwpid` of the process and lwp generating the event. The *lwp_exit*, and (process) *exit* events are delivered *before* the underlying system calls begin, while the *exec*, *fork*, and *lwp_create* events are only delivered after the relevant system calls complete successfully. The *exec* handler is passed a string that describes the command being executed. Catching the *fork* event causes the calling process to `fork(2)`, then capture the child of the controlled process using `pctx_capture()` before handing control to the *fork* handler. The process is released on return from the handler.

RETURN VALUES Upon successful completion, `pctx_set_events()` returns 0. Otherwise, the function returns -1.

EXAMPLES **EXAMPLE 1** HandleExec example.

This example captures an existing process whose process identifier is *pid*, and arranges to call the *HandleExec* routine when the process performs an `exec(2)`.

```
static void
HandleExec(pctx_t *pctx, pid_t pid, id_t lwpid, char *cmd, void *arg)
{
    (void) printf("pid %d execed '%s'\n", (int)pid, cmd);
}
```

pctx_set_events(3CPC)

EXAMPLE 1 HandleExec example. *(Continued)*

```
int
main()
{
    ...
    pctx = pctx_capture(pid, NULL, 1, NULL);
    (void) pctx_set_events(pctx,
        PCTX_SYSC_EXEC_EVENT, HandleExec,
        ...
        PCTX_NULL_EVENT);
    (void) pctx_run(pctx, 0, 0, NULL);
    pctx_release(pctx);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe
Availability	SUNWcpcu (32-bit) SUNWcpcux (64-bit)
Interface Stability	Evolving

SEE ALSO exec(2), exit(2), fork(2), vfork(2), fork1(2), _lwp_create(2), _lwp_exit(2), cpc(3CPC), proc(4), attributes(5).

NAME	picld_log – log a message in system log				
SYNOPSIS	<pre>cc [flag...] file ... -lpicltree [library ...] #include <picltree.h> void picld_log(const char *msg) ;</pre>				
DESCRIPTION	The <code>picld_log()</code> function logs the message specified in <code>msg</code> to the system log file using <code>syslog(3C)</code> . This function is used by the PICL daemon and the plug-in modules to log messages to inform users of any error or warning conditions.				
RETURN VALUES	This function does not return a value.				
ERRORS	No errors are defined.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>syslog(3C)</code> , <code>attributes(5)</code>				

picld_plugin_register(3PICLTREE)

NAME	picld_plugin_register – register plug-in with the daemon				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int picld_plugin_register(picld_plugin_reg_t *<i>regp</i>);</pre>				
DESCRIPTION	The <code>picld_plugin_register()</code> function is the function used by a plug-in module to register itself with the PICL daemon upon initialization. The plug-in provides its name and the entry points of the initialization and cleanup routines in the <i>regp</i> argument. <pre>typedef struct { int version; /* PICLD_PLUGIN_VERSION */ int critical; /* is plug-in critical? */ char *name; /* name of the plugin module */ void (*plugin_init)(void); /* init/reinit function */ void (*plugin_fini)(void); /* fini/cleanup function */ } picld_plugin_reg_t;</pre> The plug-in module also specifies whether it is a critical module for the proper system operation. The critical field in the registration information is set to <code>PICLD_PLUGIN_NON_CRITICAL</code> by plug-in modules that are not critical to system operation, and is set to <code>PICLD_PLUGIN_CRITICAL</code> by plug-in modules that are critical to the system operation. An environment control plug-in module is an example for a <code>PICLD_PLUGIN_CRITICAL</code> type of plug-in module. The PICL daemon saves the information passed during registration in <i>regp</i> in the order in which the plug-ins registered. Upon initialization, the PICL daemon invokes the <code>plugin_init()</code> routine of each of the registered plug-in modules in the order in which they registered. In their <code>plugin_init()</code> routines, the plug-in modules collect the platform configuration data and add it to the PICL tree using PICLTREE interfaces (3PICLTREE). On reinitialization, the PICL daemon invokes the <code>plugin_fini()</code> routines of the registered plug-in modules in the reverse order of registration. Then, the <code>plugin_init()</code> entry points are invoked again in the order in which the plug-ins registered.				
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a negative value is returned.				
ERRORS	<table><tr><td>PICL_NOTSUPPORTED</td><td>Version not supported</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTSUPPORTED	Version not supported	PICL_FAILURE	General system failure
PICL_NOTSUPPORTED	Version not supported				
PICL_FAILURE	General system failure				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Interface Stability	Evolving				

picld_plugin_register(3PICLTREE)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO libpicltree(3PICLTREE), attributes(5)

picl_get_first_prop(3PICL)

NAME	picl_get_first_prop, picl_get_next_prop – get a property handle of a node																
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_get_first_prop(picl_nodehdl_t <i>nodeh</i>, piclprop_hdl_t *<i>proph</i>) ; int picl_get_next_prop(picl_prophdl_t <i>proph</i>, picl_prophdl_t *<i>nextproph</i>) ;</pre>																
DESCRIPTION	The <code>picl_get_first_prop()</code> function gets the handle of the first property of the node specified by <i>nodeh</i> and copies it into the location given by <i>proph</i>. The <code>picl_get_next_prop()</code> function gets the handle of the next property after the one specified by <i>proph</i> from the property list of the node, and copies it into the location specified by <i>nextproph</i>. If there are no more properties, this function returns <code>PICL_ENDOFLIST</code>.																
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_ENDOFLIST</code> is returned to indicate that there are no more properties. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.																
ERRORS	<table> <tr> <td><code>PICL_NOTINITIALIZED</code></td><td>Session not initialized</td></tr> <tr> <td><code>PICL_NORESPONSE</code></td><td>Daemon not responding</td></tr> <tr> <td><code>PICL_NOTNODE</code></td><td>Not a node</td></tr> <tr> <td><code>PICL_NOTPROP</code></td><td>Not a property</td></tr> <tr> <td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle</td></tr> <tr> <td><code>PICL_STALEHANDLE</code></td><td>Stale handle</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> <tr> <td><code>PICL_ENDOFLIST</code></td><td>End of list</td></tr> </table>	<code>PICL_NOTINITIALIZED</code>	Session not initialized	<code>PICL_NORESPONSE</code>	Daemon not responding	<code>PICL_NOTNODE</code>	Not a node	<code>PICL_NOTPROP</code>	Not a property	<code>PICL_INVALIDHANDLE</code>	Invalid handle	<code>PICL_STALEHANDLE</code>	Stale handle	<code>PICL_FAILURE</code>	General system failure	<code>PICL_ENDOFLIST</code>	End of list
<code>PICL_NOTINITIALIZED</code>	Session not initialized																
<code>PICL_NORESPONSE</code>	Daemon not responding																
<code>PICL_NOTNODE</code>	Not a node																
<code>PICL_NOTPROP</code>	Not a property																
<code>PICL_INVALIDHANDLE</code>	Invalid handle																
<code>PICL_STALEHANDLE</code>	Stale handle																
<code>PICL_FAILURE</code>	General system failure																
<code>PICL_ENDOFLIST</code>	End of list																
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe												
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
MT-Level	MT-Safe																

`picl_get_first_prop(3PICL)`

SEE ALSO `picl_get_prop_by_name(3PICL)`, `attributes(5)`

picl_get_next_by_row(3PICL)

NAME	picl_get_next_by_row, picl_get_next_by_col – access a table property														
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_get_next_by_row(picl_prophdl_t <i>proph</i>, picl_prophdl_t *<i>colh</i>) ; int picl_get_next_by_col(picl_prophdl_t <i>proph</i>, picl_prophdl_t *<i>colh</i>) ;</pre>														
DESCRIPTION	The <code>picl_get_next_by_row()</code> function copies the handle of the property that is in the next column of the table and on the same row as the property <i>proph</i>. The handle is copied into the location given by <i>rowh</i>. The <code>picl_get_next_by_col()</code> function copies the handle of the property that is in the next row of the table and on the same column as the property <i>proph</i>. The handle is copied into the location given by <i>colh</i>. If there are no more rows or columns, this function returns the value <code>PICL_ENDOFLIST</code>.														
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.														
ERRORS	<table> <tr> <td><code>PICL_NOTINITIALIZED</code></td><td>Session not initialized</td></tr> <tr> <td><code>PICL_NORESPONSE</code></td><td>Daemon not responding</td></tr> <tr> <td><code>PICL_NOTTABLE</code></td><td>Not a table</td></tr> <tr> <td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle</td></tr> <tr> <td><code>PICL_STALEHANDLE</code></td><td>Stale handle</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> <tr> <td><code>PICL_ENDOFLIST</code></td><td>General system failure</td></tr> </table>	<code>PICL_NOTINITIALIZED</code>	Session not initialized	<code>PICL_NORESPONSE</code>	Daemon not responding	<code>PICL_NOTTABLE</code>	Not a table	<code>PICL_INVALIDHANDLE</code>	Invalid handle	<code>PICL_STALEHANDLE</code>	Stale handle	<code>PICL_FAILURE</code>	General system failure	<code>PICL_ENDOFLIST</code>	General system failure
<code>PICL_NOTINITIALIZED</code>	Session not initialized														
<code>PICL_NORESPONSE</code>	Daemon not responding														
<code>PICL_NOTTABLE</code>	Not a table														
<code>PICL_INVALIDHANDLE</code>	Invalid handle														
<code>PICL_STALEHANDLE</code>	Stale handle														
<code>PICL_FAILURE</code>	General system failure														
<code>PICL_ENDOFLIST</code>	General system failure														
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe										
ATTRIBUTE TYPE	ATTRIBUTE VALUE														
MT-Level	MT-Safe														
SEE ALSO	<code>picl_get_propval(3PICL)</code> , <code>attributes(5)</code>														

picl_get_prop_by_name(3PICL)

NAME	picl_get_prop_by_name – get the handle of the property by name																
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_get_prop_by_name(picl_nodehdl_t <i>nodeh</i>, char *<i>name</i>, picl_prophdl_t *<i>proph</i>);</pre>																
DESCRIPTION	The <code>picl_get_prop_by_name()</code> function gets the handle of the property of node <i>nodeh</i> whose name is specified in <i>name</i> . The handle is copied into the location specified by <i>proph</i> .																
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_PROPNOTFOUND is returned if the property of the specified name does not exist. PICL_RESERVEDNAME is returned if the property name specified is one of the reserved property names. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.																
ERRORS	<table> <tr> <td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr> <tr> <td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr> <tr> <td>PICL_RESERVEDNAME</td><td>Reserved property name specified</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_NOTNODE	Not a node	PICL_PROPNOTFOUND	Property not found	PICL_RESERVEDNAME	Reserved property name specified	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized																
PICL_NORESPONSE	Daemon not responding																
PICL_NOTNODE	Not a node																
PICL_PROPNOTFOUND	Property not found																
PICL_RESERVEDNAME	Reserved property name specified																
PICL_INVALIDHANDLE	Invalid handle																
PICL_STALEHANDLE	Stale handle																
PICL_FAILURE	General system failure																
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe												
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
MT-Level	MT-Safe																
SEE ALSO	<code>attributes(5)</code>																

picl_get_propinfo(3PICL)

NAME	picl_get_propinfo – get the information about a property												
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicl [<i>library...</i>] #include <picl.h> int picl_get_propinfo(picl_prophdl_t <i>proph</i>, picl_propinfo_t *<i>pinfo</i>) ;</pre>												
DESCRIPTION	The <code>picl_get_propinfo()</code> function gets the information about the property specified by handle <i>proph</i> and copies it into the location specified by <i>pinfo</i>. The property information includes the property type, access mode, size, and the name of the property as described on <code>libpicl(3PICL)</code> manual page. The maximum size of a property value is specified by <code>PICL_PROPSIZE_MAX</code>. It is currently set to 512KB.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.												
ERRORS	<table><tr><td><code>PICL_NOTINITIALIZED</code></td><td>Session not initialized</td></tr><tr><td><code>PICL_NORESPONSE</code></td><td>Daemon not responding</td></tr><tr><td><code>PICL_NOTPROP</code></td><td>Not a property</td></tr><tr><td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle specified</td></tr><tr><td><code>PICL_STALEHANDLE</code></td><td>Stale handle specific</td></tr><tr><td><code>PICL_FAILURE</code></td><td>General system failure</td></tr></table>	<code>PICL_NOTINITIALIZED</code>	Session not initialized	<code>PICL_NORESPONSE</code>	Daemon not responding	<code>PICL_NOTPROP</code>	Not a property	<code>PICL_INVALIDHANDLE</code>	Invalid handle specified	<code>PICL_STALEHANDLE</code>	Stale handle specific	<code>PICL_FAILURE</code>	General system failure
<code>PICL_NOTINITIALIZED</code>	Session not initialized												
<code>PICL_NORESPONSE</code>	Daemon not responding												
<code>PICL_NOTPROP</code>	Not a property												
<code>PICL_INVALIDHANDLE</code>	Invalid handle specified												
<code>PICL_STALEHANDLE</code>	Stale handle specific												
<code>PICL_FAILURE</code>	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe								
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
MT-Level	MT-Safe												
SEE ALSO	<code>libpicl(3PICL)</code> , <code>picl_get_propval(3PICL)</code> , <code>picl_get_propval_by_name(3PICL)</code> , <code>attributes(5)</code>												

picl_get_propinfo_by_name(3PICL)

NAME	picl_get_propinfo_by_name – get property information and handle of named property																
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_get_propinfo_by_name(picl_nodehdl_t <i>nodeh</i>, const char *<i>pname</i>, picl_propinfo_t *<i>pinfo</i>, picl_prophdl_t *<i>proph</i>);</pre>																
DESCRIPTION	The <code>picl_get_propinfo_by_name()</code> function copies the property information of the property specified by <i>pname</i> in the node <i>nodeh</i> into the location given by <i>pinfo</i> . The handle of the property is returned in the location <i>proph</i> .																
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_PROPNOTFOUND is returned if the property of the specified name does not exist. PICL_RESERVEDNAME is returned if the property name specified is one of the reserved property names. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.																
ERRORS	<table><tr><td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr><tr><td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr><tr><td>PICL_NOTNODE</td><td>Not a node</td></tr><tr><td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr><tr><td>PICL_RESERVEDNAME</td><td>Reserved property name specified</td></tr><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_NOTNODE	Not a node	PICL_PROPNOTFOUND	Property not found	PICL_RESERVEDNAME	Reserved property name specified	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized																
PICL_NORESPONSE	Daemon not responding																
PICL_NOTNODE	Not a node																
PICL_PROPNOTFOUND	Property not found																
PICL_RESERVEDNAME	Reserved property name specified																
PICL_INVALIDHANDLE	Invalid handle																
PICL_STALEHANDLE	Stale handle																
PICL_FAILURE	General system failure																
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe												
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
MT-Level	MT-Safe																
SEE ALSO	<code>picl_get_propinfo(3PICL)</code> , <code>picl_get_prop_by_name(3PICL)</code> , <code>attributes(5)</code>																

picl_get_propval(3PICL)

NAME	picl_get_propval, picl_get_propval_by_name – get the value of a property																		
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_get_propval(picl_prophdl_t <i>proph</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>) ; int picl_get_propval_by_name(picl_nodehdl_t <i>nodeh</i>, char *<i>propname</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>) ;</pre>																		
DESCRIPTION	The <code>picl_get_propval()</code> function copies the value of the property specified by the handle <i>proph</i> into the buffer location given by <i>valbuf</i>. The size of the buffer <i>valbuf</i> in bytes is specified in <i>nbytes</i>. The <code>picl_get_propval_by_name()</code> function gets the value of property named <i>propname</i> of the node specified by handle <i>nodeh</i>. The value is copied into the buffer location given by <i>valbuf</i>. The size of the buffer <i>valbuf</i> in bytes is specified in <i>nbytes</i>. The <code>picl_get_propval_by_name()</code> function is used to get a reserved property's value. An example of a reserved property is "_parent". Please refer to <code>libpicl(3PICL)</code> for a complete list of reserved property names.																		
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_PROPNOTFOUND is returned if the property of the specified name does not exist. PICL_PERMDENIED is returned if the client does not have sufficient permission to access the property. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.																		
ERRORS	<table> <tr> <td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr> <tr> <td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr> <tr> <td>PICL_PERMDENIED</td><td>Insufficient permission</td></tr> <tr> <td>PICL_VALUETOOBIG</td><td>Value too big for buffer</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_PROPNOTFOUND</td><td>Property node found</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle specified</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle specified</td></tr> </table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_PERMDENIED	Insufficient permission	PICL_VALUETOOBIG	Value too big for buffer	PICL_NOTPROP	Not a property	PICL_PROPNOTFOUND	Property node found	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle specified	PICL_STALEHANDLE	Stale handle specified
PICL_NOTINITIALIZED	Session not initialized																		
PICL_NORESPONSE	Daemon not responding																		
PICL_PERMDENIED	Insufficient permission																		
PICL_VALUETOOBIG	Value too big for buffer																		
PICL_NOTPROP	Not a property																		
PICL_PROPNOTFOUND	Property node found																		
PICL_NOTNODE	Not a node																		
PICL_INVALIDHANDLE	Invalid handle specified																		
PICL_STALEHANDLE	Stale handle specified																		

`picl_get_propval(3PICL)`

`PICL_FAILURE` General system failure

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `libpicl(3PICL)`, `picl_get_propinfo(3PICL)`, `attributes(5)`

picl_get_root(3PICL)

NAME	picl_get_root – get the root handle of the PICL tree						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicl [<i>library...</i>] #include <picl.h> int picl_get_root(picl_nodehdl_t *<i>nodehandle</i>);</pre>						
DESCRIPTION	The <code>picl_get_root()</code> function gets the handle of the root node of the PICL tree and copies it into the location given by <i>nodehandle</i> .						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.						
ERRORS	<table><tr><td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr><tr><td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized						
PICL_NORESPONSE	Daemon not responding						
PICL_FAILURE	General system failure						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	MT-Safe						
SEE ALSO	<code>picl_initialize(3PICL)</code> , <code>picl_shutdown(3PICL)</code> , <code>attributes(5)</code>						

NAME	picl_initialize – initiate a session with the PICL daemon						
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_initialize(void);</pre>						
DESCRIPTION	The <code>picl_initialize()</code> function opens the daemon door file and initiates a session with the PICL daemon running on the system.						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, this function returns a non-negative integer, <code>PICL_FAILURE</code> .						
ERRORS	<table> <tr> <td><code>PICL_NOTSUPPORTED</code></td><td>Version not supported</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> <tr> <td><code>PICL_NORESPONSE</code></td><td>Daemon not responding</td></tr> </table>	<code>PICL_NOTSUPPORTED</code>	Version not supported	<code>PICL_FAILURE</code>	General system failure	<code>PICL_NORESPONSE</code>	Daemon not responding
<code>PICL_NOTSUPPORTED</code>	Version not supported						
<code>PICL_FAILURE</code>	General system failure						
<code>PICL_NORESPONSE</code>	Daemon not responding						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	MT-Safe						
SEE ALSO	<code>picl_shutdown(3PICL)</code> , <code>attributes(5)</code>						

picl_set_propval(3PICL)

NAME	picl_set_propval, picl_set_propval_by_name – set the value of a property to the specified value																				
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_set_propval(picl_prophdl_t <i>proph</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>) ; int picl_set_propval_by_name(picl_nodehdl_t <i>nodeh</i>, const char *<i>pname</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>) ;</pre>																				
DESCRIPTION	The <code>picl_set_propval()</code> function sets the value of the property specified by the handle <i>proph</i> to the value contained in the buffer <i>valbuf</i>. The argument <i>nbytes</i> specifies the size of the buffer <i>valbuf</i>. The <code>picl_set_propval_by_name()</code> function sets the value of the property named <i>pname</i> of the node specified by the handle <i>nodeh</i> to the value contained in the buffer <i>valbuf</i>. The argument <i>nbytes</i> specifies the size of the buffer <i>valbuf</i>.																				
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_PERMDENIED is returned if the client does not have sufficient permission to access the property. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.																				
ERRORS	<table> <tr> <td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr> <tr> <td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr> <tr> <td>PICL_PERMDENIED</td><td>Insufficient permission</td></tr> <tr> <td>PICL_NOTWRITABLE</td><td>Property is read-only</td></tr> <tr> <td>PICL_VALUETOOBIG</td><td>Value too big</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle specified</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle specified</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_PERMDENIED	Insufficient permission	PICL_NOTWRITABLE	Property is read-only	PICL_VALUETOOBIG	Value too big	PICL_NOTPROP	Not a property	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle specified	PICL_STALEHANDLE	Stale handle specified	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized																				
PICL_NORESPONSE	Daemon not responding																				
PICL_PERMDENIED	Insufficient permission																				
PICL_NOTWRITABLE	Property is read-only																				
PICL_VALUETOOBIG	Value too big																				
PICL_NOTPROP	Not a property																				
PICL_NOTNODE	Not a node																				
PICL_INVALIDHANDLE	Invalid handle specified																				
PICL_STALEHANDLE	Stale handle specified																				
PICL_FAILURE	General system failure																				

`picl_set_propval(3PICL)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `attributes(5)`

picl_shutdown(3PICL)

NAME	picl_shutdown – shutdown the session with the PICL daemon				
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicl [<i>library...</i>] #include <picl.h> void picl_shutdown(void) ;</pre>				
DESCRIPTION	The <code>picl_shutdown()</code> function terminates the session with the PICL daemon and frees up any resources allocated.				
RETURN VALUES	The <code>picl_shutdown()</code> function does not return a value.				
ERRORS	<table><tr><td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTINITIALIZED	Session not initialized	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized				
PICL_FAILURE	General system failure				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>picl_initialize(3PICL)</code> , <code>attributes(5)</code>				

NAME	picl_strerror – get error message string				
SYNOPSIS	<pre>cc [flag...] file ... -lpicl [library ...] #include <picl.h> char *picl_strerror(int errnum);</pre>				
DESCRIPTION	The <code>picl_strerror()</code> function maps the error number in <i>errnum</i> to an error message string, and returns a pointer to that string. The returned string should not be overwritten.				
RETURN VALUES	The <code>picl_strerror()</code> function returns <code>NULL</code> if <i>errnum</i> is out-of-range.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>libpicl(3PICL)</code> , <code>attributes(5)</code>				

picl_wait(3PICL)

NAME	picl_wait – wait for PICL tree to refresh								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicl [<i>library...</i>] #include <picl.h> int picl_wait(int <i>to_secs</i>);</pre>								
DESCRIPTION	The <code>picl_wait()</code> function blocks the calling thread until the PICL tree is refreshed. The <i>to_secs</i> argument specifies the timeout for the call in number of seconds. A value of -1 for <i>to_secs</i> specifies no timeout.								
RETURN VALUES	The <code>picl_wait()</code> function returns 0 to indicate that PICL tree has refreshed. Otherwise, a non-negative integer is returned to indicate error.								
ERRORS	<table><tr><td>PICL_NOTINITIALIZED</td><td>Session not initialized</td></tr><tr><td>PICL_NORESPONSE</td><td>Daemon not responding</td></tr><tr><td>PICL_TIMEDOUT</td><td>Timed out waiting for refresh</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTINITIALIZED	Session not initialized	PICL_NORESPONSE	Daemon not responding	PICL_TIMEDOUT	Timed out waiting for refresh	PICL_FAILURE	General system failure
PICL_NOTINITIALIZED	Session not initialized								
PICL_NORESPONSE	Daemon not responding								
PICL_TIMEDOUT	Timed out waiting for refresh								
PICL_FAILURE	General system failure								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	MT-Safe								
SEE ALSO	<code>attributes(5)</code>								

NAME	picl_walk_tree_by_class – walk subtree by class												
SYNOPSIS	<pre>cc [flag...] file... -lpicl [library...] #include <picl.h> int picl_walk_tree_by_class(picl_nodehdl_t <i>rooth</i>, const char *<i>classname</i>, void *<i>c_args</i>, int (*<i>callback</i>)(picl_nodehdl_t <i>nodeh</i>, void *<i>c_args</i>)) ;</pre>												
DESCRIPTION	The <code>picl_walk_tree_by_class()</code> function visits all the nodes of the subtree under the node specified by <i>rooth</i>. The PICL class name of the visited node is compared with the class name specified by <i>classname</i>. If the class names match, then the callback function specified by <i>callback</i> is called with the matching node handle and the argument provided in <i>c_args</i>. If the class name specified in <i>classname</i> is NULL, then the callback function is invoked for all the nodes. The return value from the callback function is used to determine whether to continue or terminate the tree walk. The callback function returns <code>PICL_WALK_CONTINUE</code> or <code>PICL_WALK_TERMINATE</code> to continue or terminate the tree walk.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed. This error may be returned for a previously valid handle if the daemon was brought down and restarted. When this occurs a client must revalidate any saved handles.												
ERRORS	<table> <tr> <td><code>PICL_NOTINITIALIZED</code></td><td>Session not initialized</td></tr> <tr> <td><code>PICL_NORESPONSE</code></td><td>Daemon not responding</td></tr> <tr> <td><code>PICL_NOTNODE</code></td><td>Not a node</td></tr> <tr> <td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle specified</td></tr> <tr> <td><code>PICL_STALEHANDLE</code></td><td>Stale handle specified</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> </table>	<code>PICL_NOTINITIALIZED</code>	Session not initialized	<code>PICL_NORESPONSE</code>	Daemon not responding	<code>PICL_NOTNODE</code>	Not a node	<code>PICL_INVALIDHANDLE</code>	Invalid handle specified	<code>PICL_STALEHANDLE</code>	Stale handle specified	<code>PICL_FAILURE</code>	General system failure
<code>PICL_NOTINITIALIZED</code>	Session not initialized												
<code>PICL_NORESPONSE</code>	Daemon not responding												
<code>PICL_NOTNODE</code>	Not a node												
<code>PICL_INVALIDHANDLE</code>	Invalid handle specified												
<code>PICL_STALEHANDLE</code>	Stale handle specified												
<code>PICL_FAILURE</code>	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe								
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
MT-Level	MT-Safe												
SEE ALSO	<code>picl_get_propval_by_name(3PICL)</code> , <code>attributes(5)</code>												

pool_associate(3POOL)

NAME	pool_associate, pool_create, pool_destroy, pool_dissociate, pool_info, pool_query_pool_resources – resource pool manipulation functions
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpool [<i>library ...</i>] #include <pool.h> int pool_associate(pool_conf_t *conf, pool_t *pool, pool_resource_t *resource); pool_t *pool_create(pool_conf_t *conf, const char *name); int pool_destroy(pool_conf_t *conf, pool_t *pool); int pool_dissociate(pool_conf_t *conf, pool_t *pool, pool_resource_t *resource); const char *pool_info(pool_conf_t *conf, pool_t *pool, int flags); pool_resource_t *pool_query_pool_resources(pool_conf_t *conf, pool_t *pool, uint_t *nelem, pool_value_t **properties);</pre>
DESCRIPTION	These functions provide mechanisms for constructing and modifying pools entries within a target pools configuration. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_associate()</code> function associates the specified resource with <i>pool</i>. A resource can be associated with multiple pools at the same time. Any resource of this type that was formerly associated with this pool is no longer associated with the pool. The new association replaces the earlier one. The <code>pool_create()</code> function creates a new pool with the supplied name with its default properties initialized, and associated with the default resource of each type. The <code>pool_destroy</code> function() destroys the given pool association. Associated resources are not modified. The <code>pool_dissociate()</code> function removes the association between the given resource and pool. On successful completion, the pool is associated with the default resource of the same type. The <code>pool_info()</code> function returns a string describing the given pool. The string is allocated with <code>malloc(3C)</code>. The caller is responsible for freeing the returned string. If the <i>flags</i> option is non-zero, the string returned also describes the associated resources of the pool. The <code>pool_query_pool_resources()</code> function returns a null-terminated array of resources currently associated with the pool. The return value must be freed by the caller. The <i>nelem</i> argument is set to be the length of the array returned.
RETURN VALUES	Upon successful completion, <code>pool_create()</code> returns a new initialized pool. Otherwise it returns NULL and <code>pool_error(3POOL)</code> returns the pool-specific error value.

pool_associate(3POOL)

Upon successful completion, pool_associate(), pool_destroy(), and pool_dissociate() return 0. Otherwise, they return -1 and pool_error() returns the pool-specific error value.

Upon successful completion, pool_info() returns a string describing the given pool. Otherwise it returns NULL and pool_error() returns the pool-specific error value.

Upon successful completion, pool_query_pool_resources() returns a null-terminated array of resources. Otherwise it returns NULL and pool_error() returns the pool-specific error value.

ERRORS

The pool_create() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID or <i>name</i> is already in use.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.
POE_INVALID_CONF	The pool element could not be created because the configuration would be invalid.
POE_PUTPROP	One of the supplied properties could not be set.

The pool_destroy() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.

The pool_associate() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID or the parameters are supplied from a different configuration.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The pool_disassociate() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID or the parameters are supplied from a different configuration.
POE_INVALID_CONF	No resources could be located for the supplied configuration or the supplied configuration is not valid (for example, more than one default for a resource type was found.)
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The pool_info() function will fail if:

pool_associate(3POOL)

POE_BADPARAM	The supplied configuration's status is not POF_VALID or the <i>flags</i> parameter is neither 0 or 1.
POE_INVALID_CONF	The configuration is invalid.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The `pool_query_pool_resources()` function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.
POE_INVALID_CONF	The configuration is invalid.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

USAGE Pool names are unique across pools in a given configuration file. It is an error to attempt to create a pool with a name that is currently used by another pool within the same configuration.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO `libpool(3LIB)`, `pool_error(3POOL)`, `attributes(5)`

NAME	pool_component_info, pool_get_owning_resource – resource pool component functions								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpool [<i>library...</i>] #include <pool.h> const char *pool_component_info(pool_conf_t *<i>conf</i>, pool_component_t *<i>component</i>, int <i>flags</i>); pool_resource_t *pool_get_owning_resource(pool_conf_t *<i>conf</i>, pool_component_t *<i>component</i>);</pre>								
DESCRIPTION	Certain resources, such as processor sets, are composed of resource components. Informational and ownership attributes of resource components are made available with the <code>pool_component_info()</code> and <code>pool_get_owning_resource()</code> functions. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_component_info()</code> function returns a string describing <i>component</i>. The string is allocated with <code>malloc(3C)</code>. The caller is responsible for freeing the returned string. The <i>flags</i> argument is ignored. The <code>pool_get_owning_resource()</code> function returns the resource currently containing <i>component</i>. Every component is contained by a resource.								
RETURN VALUES	Upon successful completion, <code>pool_component_info()</code> returns a string. Otherwise it returns NULL and <code>pool_error(3POOL)</code> returns the pool-specific error value. Upon successful completion, <code>pool_get_owning_resource()</code> returns the owning resource. Otherwise it returns NULL and <code>pool_error()</code> returns the pool-specific error value.								
ERRORS	The <code>pool_component_info()</code> function will fail if: <table> <tr> <td>POE_BADPARAM</td><td>The supplied configuration's status is not POE_VALID or the <i>flags</i> parameter is neither 0 or 1.</td></tr> <tr> <td>POE_INVALID_CONF</td><td>The configuration is invalid.</td></tr> <tr> <td>POE_SYSTEM</td><td>A system error has occurred. Check the system error code for more details.</td></tr> </table> The <code>pool_get_owning_resource()</code> function will fail if: <table> <tr> <td>POE_BADPARAM</td><td>The supplied configuration's status is not POE_VALID.</td></tr> </table>	POE_BADPARAM	The supplied configuration's status is not POE_VALID or the <i>flags</i> parameter is neither 0 or 1.	POE_INVALID_CONF	The configuration is invalid.	POE_SYSTEM	A system error has occurred. Check the system error code for more details.	POE_BADPARAM	The supplied configuration's status is not POE_VALID.
POE_BADPARAM	The supplied configuration's status is not POE_VALID or the <i>flags</i> parameter is neither 0 or 1.								
POE_INVALID_CONF	The configuration is invalid.								
POE_SYSTEM	A system error has occurred. Check the system error code for more details.								
POE_BADPARAM	The supplied configuration's status is not POE_VALID.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>CSI</td><td>Enabled</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	CSI	Enabled				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
CSI	Enabled								

pool_component_info(3POOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

pool_component_to_elem(3POOL)

NAME	pool_component_to_elem, pool_to_elem, pool_conf_to_elem, pool_resource_to_elem – resource pool element-related functions								
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> pool_elem_t *pool_component_to_elem(pool_conf_t *conf, pool_component_t *component); pool_elem_t *pool_conf_to_elem(pool_conf_t *conf); pool_elem_t *pool_resource_to_elem(pool_conf_t *conf, pool_resource_t *resource); pool_elem_t *pool_to_elem(pool_conf_t *conf, pool_t *pool);</pre>								
DESCRIPTION	A pool element, as represented by a <code>pool_elem_t</code> , is a common abstraction for any <code>libpool</code> entity that contains properties. All such types can be converted to the opaque <code>pool_elem_t</code> type using the appropriate conversion functions prototyped above. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies.								
RETURN VALUES	Upon successful completion, these functions return a <code>pool_elem_t</code> corresponding to the argument passed in. Otherwise they return NULL and <code>pool_error(3POOL)</code> returns the pool-specific error value.								
ERRORS	These function will fail if: POE_BADPARAM The supplied configuration's status is not POF_VALID.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>CSI</td><td>Enabled</td></tr><tr><td>Interface Stability</td><td>Unstable</td></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	CSI	Enabled	Interface Stability	Unstable	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
CSI	Enabled								
Interface Stability	Unstable								
MT-Level	Unsafe								
SEE ALSO	<code>libpool(3LIB)</code> , <code>pool_error(3POOL)</code> , <code>attributes(5)</code>								

pool_conf_alloc(3POOL)

NAME	pool_conf_alloc, pool_conf_close, pool_conf_commit, pool_conf_export, pool_conf_free, pool_conf_info, pool_conf_location, pool_conf_open, pool_conf_remove, pool_conf_rollback, pool_conf_status, pool_conf_validate – manipulate resource pool configurations
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpool [<i>library ...</i>] #include <pool.h> pool_conf_t *pool_conf_alloc(void); int pool_conf_close(pool_conf_t *conf); int pool_conf_commit(pool_conf_t *conf, int active); int pool_conf_export(pool_conf_t *conf, const char *location, pool_export_format_t format); void pool_conf_free(pool_conf_t *conf); char *pool_conf_info(const pool_conf_t *conf, int flags); const char *pool_conf_location(pool_conf_t *conf); int pool_conf_open(pool_conf_t *conf, const char *location, int flags); int pool_conf_remove(pool_conf_t *conf); int pool_conf_rollback(pool_conf_t *conf); pool_conf_state_t pool_conf_status(const pool_conf_t *conf); int pool_conf_validate(pool_conf_t *conf, pool_valid_level_t level);</pre>
DESCRIPTION	These functions enable the access and creation of configuration files associated with the pools facility. Since the pool configuration is an opaque type, an initial configuration is obtained with <code>pool_conf_alloc()</code> and released with <code>pool_conf_free()</code> when the configuration is no longer of interest. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_conf_close()</code> function closes the given configuration, releasing associated resources. The <code>pool_conf_commit()</code> function commits changes made to the given <code>pool_conf_t</code> to permanent storage. If the <i>active</i> flag is non-zero, the state of the system will be configured to match that described in the supplied <code>pool_conf_t</code>. If configuring the system fails, <code>pool_conf_commit()</code> will attempt to restore the system to its previous state. The <code>pool_conf_export()</code> function saves the given configuration to the specified location. The only currently supported value of <i>format</i> is <code>POX_NATIVE</code>, which is the format native to <code>libpool</code>, the output of which can be used as input to <code>pool_conf_open()</code>.

pool_conf_alloc(3POOL)

The `pool_conf_info()` function returns a string describing the entire configuration. The string is allocated with `malloc(3C)`. The caller is responsible for freeing the returned string. If the `flags` option is non-zero, the string returned also describes the sub-elements (if any) contained in the configuration.

The `pool_conf_location()` function returns the location string provided to `pool_conf_open()` for the given `pool_conf_t`.

The `pool_conf_open()` function creates a `pool_conf_t` given a location at which the configuration is stored. The valid flags are a bitmap of the following:

<code>PO_RDONLY</code>	Open for reading only.
<code>PO_RDWR</code>	Open read-write.
<code>PO_CREAT</code>	Create a configuration at the given location if it does not exist. If it does, truncate it.
<code>PO_DISCO</code>	Perform 'discovery'. This option only makes sense when used in conjunction with <code>PO_CREAT</code> , and causes the returned <code>pool_conf_t</code> to contain the resources and components currently active on the system.
<code>PO_UPDATE</code>	Use when opening the dynamic state file, which is the configuration at <code>pool_dynamic_location(3POOL)</code> , to ensure that the contents of the dynamic state file are updated to represent the current state of the system.

The `pool_conf_remove()` function removes the configuration's permanent storage. If the configuration is still open, it is first closed.

The `pool_conf_rollback()` function restores the configuration state to that held in the configuration's permanent storage. This will either be the state last successfully committed (using `pool_conf_commit()`) or the state when the configuration was opened if there have been no successfully committed changes since then.

The `pool_conf_status()` function returns the status of a configuration, which can be one of the following values:

<code>POF_INVALID</code>	The configuration is not in a suitable state for use.
<code>POF_VALID</code>	The configuration is in a suitable state for use.

The `pool_conf_validate()` function checks the validity of the contents of the given configuration. The validation can be at several (increasing) levels of strictness:

<code>POV_LOOSE</code>	Performs basic internal syntax validation.
<code>POV_STRICT</code>	Performs a more thorough syntax validation and internal consistency checks.
<code>POV_RUNTIME</code>	Performs an estimate of whether attempting to commit the given configuration on the system would succeed or fail. It is optimistic

pool_conf_alloc(3POOL)

in that a successful validation does not guarantee a subsequent commit operation will be successful; it is conservative in that a failed validation indicates that a subsequent commit operation on the current system will always fail.

RETURN VALUES

Upon successful completion, `pool_conf_alloc()` returns an initialized `pool_conf_t` pointer. Otherwise it returns `NULL` and `pool_error(3POOL)` returns the pool-specific error value.

Upon successful completion, `pool_conf_close()`, `pool_conf_commit()`, `pool_conf_export()`, `pool_conf_open()`, `pool_conf_remove()`, `pool_conf_rollback()`, and `pool_conf_validate()` return 0. Otherwise they return -1 and `pool_error()` returns the pool-specific error value.

The `pool_conf_status()` function returns either `POF_INVALID` or `POF_VALID`.

ERRORS

The `pool_conf_alloc()` function will fail if:

<code>POE_SYSTEM</code>	There is not enough memory available to allocate the configuration. Check <code>errno</code> for the specific system error code.
<code>POE_INVALID_CONF</code>	The configuration is invalid.

The `pool_conf_close()` function will fail if:

<code>POE_BADPARAM</code>	The supplied configuration's status is not <code>POF_VALID</code> .
<code>POE_SYSTEM</code>	The configuration's permanent store cannot be closed. Check <code>errno</code> for the specific system error code.

The `pool_conf_commit()` function will fail if:

<code>POE_BADPARAM</code>	The supplied configuration's status is not <code>POF_VALID</code> or the active flag is non-zero and the system could not be modified.
<code>POE_SYSTEM</code>	The permanent store could not be updated. Check <code>errno</code> for the specific system error code.
<code>POE_INVALID_CONF</code>	The configuration is not valid for this system.
<code>POE_NOTSUP</code>	The configuration was not opened for update.
<code>POE_DATASTORE</code>	The update of the permanent store has failed and the contents could be corrupted. Check for a <code>.bak</code> file at the datastore location if manual recovery is required.

The `pool_conf_export()` function will fail if:

<code>POE_BADPARAM</code>	The supplied configuration's status is not <code>POF_VALID</code> or the requested export format is not supported.

pool_conf_alloc(3POOL)

POE_DATASTORE	The creation of the export file failed. A file might have been created at the specified location but the contents of the file might not be correct.

The pool_conf_info() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID or <i>flags</i> is neither 0 nor 1.

POE_SYSTEM	There is not enough memory available to allocate the buffer used to build the information string. Check <code>errno</code> for the specific system error code.

POE_INVALID_CONF	The configuration is invalid.

The pool_conf_location() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.

The pool_conf_open() function will fail if:

POE_BADPARAM	The supplied configuration's status is already POF_VALID.

POE_SYSTEM	There is not enough memory available to store the supplied location. Check <code>errno</code> for the specific system error code.

POE_INVALID_CONF	The configuration to be opened is at <code>pool_dynamic_location(3POOL)</code> and the configuration is not valid for this system.

The pool_conf_remove() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.

POE_SYSTEM	The configuration's permanent storage could not be removed. Check <code>errno</code> for the specific system error code.

The pool_conf_rollback() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.

POE_SYSTEM	The permanent store could not be accessed. Check <code>errno</code> for the specific system error code.

The pool_conf_validate() function will fail if:

POE_BADPARAM	The supplied configuration's status is not POF_VALID.

POE_INVALID_CONF	The configuration is invalid.

pool_conf_alloc(3POOL)

EXAMPLES

EXAMPLE 1 Create the configuration at the specified location.

```
#include <pool.h>
#include <stdio.h>

...

pool_conf_t *pool_conf;
pool_conf = pool_conf_alloc();
char *input_location = "/tmp/poolconf.example";

if (pool_conf_open(pool_conf, input_location, PO_RDONLY) < 0) {
    fprintf(stderr, 'Config make from %s failed\\B{\\n', input_location);
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO

libpool(3LIB), pool_error(3POOL), attributes(5)

pool_dynamic_location(3POOL)

NAME	pool_dynamic_location, pool_static_location, pool_version – resource pool framework functions
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> const char *pool_dynamic_location(void); const char *pool_static_location(void); uint_t pool_version(uint_t ver);</pre>
DESCRIPTION	The pool_dynamic_location() function returns the location used by the pools framework to store the dynamic configuration. The pool_static_location() function returns the location used by the pools framework to store the default configuration used for pools framework instantiation. The pool_version() function may be used to enquire about the version number of the library by specifying POOL_VER_NONE. The current (most capable) version is POOL_VER_CURRENT. The user can set the version used by the library by specifying the required version number. If this is not possible, the version returned will be POOL_VER_NONE.
RETURN VALUES	The pool_dynamic_location() function returns the location used by the pools framework to store the dynamic configuration. The pool_static_location() function returns the location used by the pools framework to store the default configuration used for pools framework instantiation. The pool_version() function returns the version number of the library or POOL_VER_NONE.
ERRORS	No errors are defined.
EXAMPLES	EXAMPLE 1 Get the static location used by the pools framework. <pre>#include sys/types.h> #include <pool.h> #include <unistd.h> ... const char *location = pool_dynamic_location(); pid = getpid(); ... (void) fprintf(stderr, "pool dynamic location is \\%s\\B{\\}n", location);</pre>

pool_dynamic_location(3POOL)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

NAME	pool_error, pool_strerror – error interface to resource pools library																		
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpool [<i>library...</i>] #include <pool.h> int pool_error(void); const char *pool_strerror(int <i>perr</i>);</pre>																		
DESCRIPTION	The <code>pool_error()</code> function returns the error value of the last failure recorded by the invocation of one of the functions of the resource pool configuration library, <code>libpool</code>. The <code>pool_strerror()</code> function returns a descriptive null-terminated string for each of the valid pool error codes. The following error codes can be returned by <code>pool_error()</code>:																		
RETURN VALUES	The <code>pool_error()</code> function returns the current pool error value for the calling thread from among the following: <table> <tr> <td>POE_BADPARAM</td><td>A bad parameter was supplied.</td></tr> <tr> <td>POE_BAD_PROP_TYPE</td><td>An incorrect property type was submitted or encountered during the pool operation.</td></tr> <tr> <td>POE_DATASTORE</td><td>An error occurred within permanent storage.</td></tr> <tr> <td>POE_INVALID_CONF</td><td>The pool configuration presented for the operation is invalid.</td></tr> <tr> <td>POE_INVALID_SEARCH</td><td>A query whose outcome set was empty was attempted.</td></tr> <tr> <td>POE_NOTSUP</td><td>An unsupported operation was attempted.</td></tr> <tr> <td>POE_PUTPROP</td><td>An attempt to write a read-only property was made.</td></tr> <tr> <td>POE_OK</td><td>The previous pool operation succeeded.</td></tr> <tr> <td>POE_SYSTEM</td><td>An underlying system call or library function failed; <code>errno(3C)</code> is preserved where possible.</td></tr> </table> The <code>pool_strerror()</code> function returns a pointer to the string corresponding to the requested error value. If the error value has no corresponding string, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.	POE_BADPARAM	A bad parameter was supplied.	POE_BAD_PROP_TYPE	An incorrect property type was submitted or encountered during the pool operation.	POE_DATASTORE	An error occurred within permanent storage.	POE_INVALID_CONF	The pool configuration presented for the operation is invalid.	POE_INVALID_SEARCH	A query whose outcome set was empty was attempted.	POE_NOTSUP	An unsupported operation was attempted.	POE_PUTPROP	An attempt to write a read-only property was made.	POE_OK	The previous pool operation succeeded.	POE_SYSTEM	An underlying system call or library function failed; <code>errno(3C)</code> is preserved where possible.
POE_BADPARAM	A bad parameter was supplied.																		
POE_BAD_PROP_TYPE	An incorrect property type was submitted or encountered during the pool operation.																		
POE_DATASTORE	An error occurred within permanent storage.																		
POE_INVALID_CONF	The pool configuration presented for the operation is invalid.																		
POE_INVALID_SEARCH	A query whose outcome set was empty was attempted.																		
POE_NOTSUP	An unsupported operation was attempted.																		
POE_PUTPROP	An attempt to write a read-only property was made.																		
POE_OK	The previous pool operation succeeded.																		
POE_SYSTEM	An underlying system call or library function failed; <code>errno(3C)</code> is preserved where possible.																		
ERRORS	The <code>pool_strerror()</code> function will fail if: <table> <tr> <td>ESRCH</td><td>The specified error value is not defined by the pools error facility.</td></tr> </table>	ESRCH	The specified error value is not defined by the pools error facility.																
ESRCH	The specified error value is not defined by the pools error facility.																		
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:																		

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled

pool_error(3POOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO [errno\(3C\)](#), [libpool\(3LIB\)](#), [pool_error\(3POOL\)](#), [attributes\(5\)](#)

NAME	pool_get_binding, pool_set_binding, pool_get_resource_binding – set and query process to resource pool bindings						
SYNOPSIS	<pre>cc [flag...] file... -lpool [library ...] #include <pool.h> char *pool_get_binding(pid_t pid); int pool_set_binding(const char *pool, idtype_t idtype, id_t id); char *pool_get_resource_binding(const char *type, pid_t pid);</pre>						
DESCRIPTION	The <code>pool_get_binding()</code> function returns the name of the pool on the running system that contains the set of resources to which the given process is bound. If no such pool exists on the system or the search returns more than one pool (since the set of resources is referred to by more than one pool), <code>NULL</code> is returned and the pool error value is set to <code>POE_INVALID_SEARCH</code>. It is possible that one of the resources to which the given process is bound is not associated with a pool. This could occur if a processor set was created with one of the <code>pset_()</code> functions and the process was then bound to that set. It could also occur if the process was bound to a resource set not currently associated with a pool, since resources can exist that are not associated with a pool. The <code>pool_set_binding()</code> function binds the processes matching <i>idtype</i> and <i>id</i> to the resources associated with <i>pool</i> on the running system. This function requires the privilege required by the underlying resource types referenced by the pool; generally, this requirement is equivalent to requiring superuser privilege. The <i>idtype</i> parameter can be of the following types: <table> <tr> <td><code>P_PID</code></td><td>The <i>id</i> parameter is a pid.</td></tr> <tr> <td><code>P_TASKID</code></td><td>The <i>id</i> parameter is a taskid.</td></tr> <tr> <td><code>P_PROJID</code></td><td>The <i>id</i> parameter is a project ID. All currently running processes belonging to the given project will be bound to the pool's resources.</td></tr> </table> The <code>pool_get_resource_binding()</code> function returns the name of the resource of the supplied type to which the supplied process is bound.	<code>P_PID</code>	The <i>id</i> parameter is a pid.	<code>P_TASKID</code>	The <i>id</i> parameter is a taskid.	<code>P_PROJID</code>	The <i>id</i> parameter is a project ID. All currently running processes belonging to the given project will be bound to the pool's resources.
<code>P_PID</code>	The <i>id</i> parameter is a pid.						
<code>P_TASKID</code>	The <i>id</i> parameter is a taskid.						
<code>P_PROJID</code>	The <i>id</i> parameter is a project ID. All currently running processes belonging to the given project will be bound to the pool's resources.						
RETURN VALUES	Upon successful completion, <code>pool_get_binding()</code> returns the name of the pool to which the process is bound. Otherwise it returns <code>NULL</code> and <code>pool_error(3POOL)</code> returns the pool-specific error value. Upon successful completion, <code>pool_set_binding()</code> returns <code>PO_SUCCESS</code>. Otherwise, it returns <code>PO_FAIL</code> and <code>pool_error()</code> returns the pool-specific error value. Upon successful completion, <code>pool_get_resource_binding()</code> returns the name of the resource of the specified type to which the process is bound. Otherwise it returns <code>NULL</code> and <code>pool_error()</code> returns the pool-specific error value.						

pool_get_binding(3POOL)

ERRORS

The pool_get_binding() function will fail if:

POE_INVALID_CONF	The configuration is invalid.
POE_INVALID_SEARCH	It is not possible to determine the binding for this target due to the overlapping nature of the pools configured for this system, or the pool could not be located.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The pool_set_binding() function will fail if:

POE_BADPARAM	The pool could not be found.
POE_INVALID_CONF	The configuration is invalid.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The pool_get_resource_binding() function will fail if:

POE_INVALID_CONF	The configuration is invalid.
POE_INVALID_SEARCH	The target is not bound to a resource of the specified type.
POE_SYSTEM	A system error has occurred. Check the system error code for more details.

EXAMPLES

EXAMPLE 1 Bind the current process to the pool named "target".

```
#include <sys/types.h>
#include <pool.h>
#include <unistd.h>

...

id_t pid = getpid();

...

if (pool_set_binding("target", P_PID, pid) == PO_FAIL) {{
    (void) fprintf(stderr, "pool binding failed (\\%d)\\B{\\n",
        pool_error());
}}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled

`pool_get_binding(3POOL)`

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO `libpool(3LIB)`, `pool_error(3POOL)`, `attributes(5)`

pool_get_pool(3POOL)

NAME	pool_get_pool, pool_get_resource, pool_query_components, pool_query_pools, pool_query_resources – retrieve resource pool configuration elements
SYNOPSIS	<pre>cc [flag]... file... -lpool [library...] #include <pool.h> pool_t *pool_get_pool(pool_conf_t *conf, const char *name); pool_resource_t *pool_get_resource(pool_conf_t *conf, const char *type, const char *name); pool_component_t **pool_query_components(pool_conf_t *conf, uint_t *nelem, pool_value_t **props); pool_t **pool_query_pools(pool_conf_t *conf, uint_t *nelem, pool_value_t **props); pool_component_t **pool_query_resources(pool_conf_t *conf, uint_t *nelem, pool_value_t **props);</pre>
DESCRIPTION	These functions provide a means for querying the contents of the specified configuration. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_get_pool()</code> function returns the pool with the given name from the provided configuration. The <code>pool_get_resource()</code> function returns the resource with the given name and type from the provided configuration. The <code>pool_query_components()</code> function retrieves all resource components that match the given list of properties. If the list of properties is NULL, all components are returned. The number of elements returned is stored in the location pointed to by <i>nelem</i>. The value returned by <code>pool_query_components()</code> is allocated with <code>malloc(3C)</code> and must be explicitly freed. The <code>pool_query_pools()</code> function behaves similarly to <code>pool_query_components()</code> and returns the list of pools that match the given list of properties. The value returned must be freed by the caller. The <code>pool_query_resources()</code> function similarly returns the list of resources that match the given list of properties. The return value must be freed by the caller.
RETURN VALUES	The <code>pool_get_pool()</code> and <code>pool_get_resource()</code> functions return the matching pool and resource, respectively. Otherwise, they return -1 and <code>pool_error(3POOL)</code> returns the pool-specific error value. The <code>pool_query_components()</code>, <code>pool_query_pools()</code>, and <code>pool_query_resources()</code> functions return a null-terminated array of components, pools, and resources, respectively. If the query was unsuccessful or there were no matches, NULL is returned and <code>pool_error()</code> returns the pool-specific error value.
ERRORS	The <code>pool_get_pool()</code> will fail if:

pool_get_pool(3POOL)

POE_BADPARAM The supplied configuration's status is not POF_VALID.

The pool_get_resource() will fail if:

POE_BADPARAM The supplied configuration's status is not POF_VALID.

POE_SYSTEM There is not enough memory available to allocate working buffers. Check errno for the specific system error code.

The pool_query_components(), pool_query_pools(), and pool_query_resources() will fail if:

POE_BADPARAM The supplied configuration's status is not POF_VALID.

POE_INVALID_CONF The query generated results that were not of the correct type. The configuration is invalid.

POE_SYSTEM There is not enough memory available to allocate working buffers. Check errno for the specific system error code.

EXAMPLES

EXAMPLE 1 Retrieve the pool named "foo" from a given configuration.

```
#include <pool.h>
#include <stdio.h>

...

pool_conf_t *conf;
pool_t *pool;

...

if ((pool = pool_get_pool(conf, "foo")) == NULL) {
    (void) fprintf(stderr, "Cannot retrieve pool named
    'foo'\n");
    ...
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO

libpool(3LIB), pool_error(3POOL), attributes(5)

pool_get_property(3POOL)

NAME	pool_get_property, pool_put_property, pool_rm_property, pool_walk_properties – resource pool element property manipulation										
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> pool_value_class_t pool_get_property(pool_conf_t *conf, const pool_elem_t *elem, const char *name, pool_value_t *property); int pool_put_property(pool_conf_t *conf, pool_elem_t *elem, const char *name, const pool_value_t *value); int pool_rm_property(pool_conf_t *conf, pool_elem_t *elem, const char *name); int pool_walk_properties(pool_conf_t *conf, pool_elem_t *elem, void *arg, int (*callback)(pool_conf_t *, pool_elem_t *, const char *, pool_value_t *, void *));</pre>										
DESCRIPTION	The various pool types are converted to the common pool element type (<code>pool_elem_t</code>) before property manipulation. A <code>pool_value_t</code> is an opaque type that contains a property value of one of the following types: <table> <tr> <td>POC_UINT</td><td>unsigned 64-bit integer</td></tr> <tr> <td>POC_INT</td><td>signed 64-bit integer</td></tr> <tr> <td>POC_DOUBLE</td><td>signed double-precision floating point value</td></tr> <tr> <td>POC_BOOL</td><td>boolean value: 0 is false, non-zero is true</td></tr> <tr> <td>POC_STRING</td><td>null-terminated string of characters</td></tr> </table> The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_get_property()</code> function attempts to retrieve the value of the named property from the element. If the property is not found or an error occurs, the value <code>POC_INVALID</code> is returned to indicate error. Otherwise the type of the value retrieved is returned. The <code>pool_put_property()</code> function attempts to set the named property on the element to the specified value. Attempting to set a property that does not currently exist on the element will cause the property with the given name and value to be created on the element and will not cause an error. An attempt to overwrite an existing property with a new property of a different type is an error. The <code>pool_rm_property()</code> function attempts to remove the named property from the element. If the property does not exist or is not removable, -1 is returned and <code>pool_error(3POOL)</code> reports an error of <code>POE_PUTPROP</code>.	POC_UINT	unsigned 64-bit integer	POC_INT	signed 64-bit integer	POC_DOUBLE	signed double-precision floating point value	POC_BOOL	boolean value: 0 is false, non-zero is true	POC_STRING	null-terminated string of characters
POC_UINT	unsigned 64-bit integer										
POC_INT	signed 64-bit integer										
POC_DOUBLE	signed double-precision floating point value										
POC_BOOL	boolean value: 0 is false, non-zero is true										
POC_STRING	null-terminated string of characters										

pool_get_property(3POOL)

The `pool_walk_properties()` function invokes *callback* on all properties defined for the given element. The *callback* is called with the element itself, the name of the property, the value of the property, and the caller-provided opaque argument.

A number of special properties are reserved for internal use and cannot be set or removed. Attempting to do so will fail. These properties are documented on the `libpool(3LIB)` manual page.

RETURN VALUES

Upon successful completion, `pool_get_property()` returns the type of the property. Otherwise it returns `POC_INVALID` and `pool_error()` returns the pool-specific error value.

Upon successful completion, `pool_put_property()`, `pool_rm_property()`, and `pool_walk_properties()` return 0. Otherwise they return -1 and `pool_error()` returns the pool-specific error value.

ERRORS

The `pool_get_property()` function will fail if:

`POE_BADPARAM` The supplied configuration's status is not `POF_VALID`, the supplied *conf* does not contain the supplied *elem*, or the property is restricted and cannot be accessed by the library.

`POE_SYSTEM` A system error has occurred. Check the system error code for more details.

The `pool_put_property()` function will fail if:

`POE_BADPARAM` The supplied configuration's status is not `POF_VALID`, the supplied *conf* does not contain the supplied *elem*, the property name is not in the correct format, or the property already exists and the supplied type does not match the existing type.

`POE_SYSTEM` A system error has occurred. Check the system error code for more details.

`POE_PUTPROP` The property name is reserved by `libpool` and not available for use.

`POE_INVALID_CONF` The configuration is invalid.

The `pool_rm_property()` function will fail if:

`POE_BADPARAM` The supplied configuration's status is not `POF_VALID`, the supplied *conf* does not contain the supplied *elem*, or the property is reserved by `libpool` and cannot be removed.

`POE_SYSTEM` A system error has occurred. Check the system error code for more details.

pool_get_property(3POOL)

POE_PUTPROP The property name is reserved by libpool and not available for use.

The pool_walk_properties() function will fail if:

POE_BADPARAM The supplied configuration's status is not POF_VALID.

POE_SYSTEM A system error has occurred. Check the system error code for more details.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

pool_resource_create(3POOL)

NAME	pool_resource_create, pool_resource_destroy, pool_resource_info, pool_query_resource_components, pool_resource_transfer, pool_resource_xtransfer – resource pool resource manipulation functions
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> pool_resource_t *pool_resource_create(pool_conf_t *conf, const char *type, const char *name); int pool_resource_destroy(pool_conf_t *conf, pool_resource_t *resource); const char *pool_resource_info(pool_conf_t *conf, pool_resource_t *resource, int flags); pool_component_t **pool_query_resource_components(pool_conf_t *conf, pool_resource_t *resource, uint_t *nelem, pool_value_t **props); int pool_resource_transfer(pool_conf_t *conf, pool_resource_t *source, pool_resource_t *target, uint64_t size); int pool_resource_xtransfer(pool_conf_t *conf, pool_resource_t *source, pool_resource_t *target, pool_component_t **components);</pre>
DESCRIPTION	The <code>pool_resource_create()</code> function creates and returns a new resource of the given <i>name</i> and <i>type</i> in the provided configuration. If there is already a resource of the given name, the operation will fail. The <code>pool_resource_destroy()</code> function removes the specified <i>resource</i> from its configuration file. The <code>pool_resource_info()</code> function returns a string describing the given <i>resource</i>. The string is allocated with <code>malloc(3C)</code>. The caller is responsible for freeing the returned string. If the <i>flags</i> argument is non-zero, the string returned also describes the components (if any) contained in the resource. The <code>pool_query_resource_components()</code> function returns a null-terminated array of the components (if any) that comprise the given resource. The <code>pool_resource_transfer()</code> function transfers <i>size</i> basic units from the <i>source</i> resource to the <i>target</i>. Both resources must be of the same type for the operation to succeed. Transferring units in this manner translates to lowering the size property of the source resource by the specified <i>size</i>, while increasing the size property of the target by the same. The <code>pool_resource_xtransfer()</code> function transfers the specific <i>components</i> from the <i>source</i> resource to the <i>target</i>. Both resources must be of the same type, and of a type that contains components (such as processor sets). The <i>components</i> argument is a null-terminated list of <code>pool_component_t</code>.

pool_resource_create(3POOL)

	The <i>conf</i> argument for each function refers to the target configuration to which the operation applies.	
RETURN VALUES	Upon successful completion, pool_resource_create() returns a new pool_resource_t with default properties initialized. Otherwise, NULL is returned and pool_error(3POOL) returns the pool-specific error value.	
	Upon successful completion, pool_resource_destroy() returns 0. Otherwise, -1 is returned and pool_error() returns the pool-specific error value.	
	Upon successful completion, pool_resource_info() returns a string describing the given resource (and optionally its components). Otherwise, NULL is returned and pool_error() returns the pool-specific error value.	
	Upon successful completion, pool_query_resource_components() returns a null-terminated array of pool_component_t * that match the provided null-terminated property list and are contained in the given resource. Otherwise, NULL is returned and pool_error() returns the pool-specific error value.	
	Upon successful completion, pool_resource_transfer() and pool_resource_xtransfer() return 0. Otherwise -1 is returned and pool_error() returns the pool-specific error value.	
ERRORS	The pool_resource_create() function will fail if:	
	POE_BADPARAM	The supplied configuration's status is not POF_VALID or <i>name</i> is in use for this resource type.
	POE_INVALID_CONF	The resource element could not be created because the configuration would be invalid.
	POE_PUTPROP	One of the supplied properties could not be set.
	POE_SYSTEM	A system error has occurred. Check the system error code for more details.
	The pool_resource_destroy() function will fail if:	
	POE_BADPARAM	The supplied configuration's status is not POF_VALID.
	The pool_resource_info() function will fail if:	
	POE_BADPARAM	The supplied configuration's status is not POF_VALID or the <i>flags</i> paramter is neither 0 nor 1.
	POE_INVALID_CONF	The configuration is invalid.
	POE_SYSTEM	A system error has occurred. Check the system error code for more details.
	The pool_query_resource_components() function will fail if:	
	POE_BADPARAM	The supplied configuration's status is not POF_VALID.

pool_resource_create(3POOL)

POE_INVALID_CONF The configuration is invalid.

POE_SYSTEM A system error has occurred. Check the system error code for more details.

The pool_resource_transfer() function will fail if:

POE_BADPARAM The supplied configuration's status is not POF_VALID, the two resources are not of the same type, or the transfer would cause either of the resources to exceed their min and max properties.

POE_SYSTEM A system error has occurred. Check the system error code for more details.

The pool_resource_xtransfer() function will fail if:

POE_BADPARAM The supplied configuration's status is not POF_VALID, the two resources are not of the same type, or the supplied resources do not belong to the source.

POE_INVALID_CONF The transfer operation failed and the configuration may be invalid.

POE_SYSTEM A system error has occurred. Check the system error code for more details.

EXAMPLES

EXAMPLE 1 Create a new resource of type pset named foo.

```
#include <pool.h>
#include <stdio.h>

...

pool_conf_t *conf;
pool_resource_t *resource;
...

if ((resource = pool_resource_create(conf, "pset", "foo")) == NULL) {
    (void) fprintf(stderr, "Cannot create resource\\B{n");
    ...
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

pool_resource_create(3POOL)

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

pool_value_alloc(3POOL)

NAME	pool_value_alloc, pool_value_free, pool_value_get_bool, pool_value_get_double, pool_value_get_int64, pool_value_get_name, pool_value_get_string, pool_value_get_type, pool_value_get_uint64, pool_value_set_bool, pool_value_set_double, pool_value_set_int64, pool_value_set_name, pool_value_set_string, pool_value_set_uint64 – resource pool property value manipulation functions
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> pool_value_t *pool_value_alloc(void); void pool_value_free(pool_value_t *value); pool_value_class_t pool_value_get_type(const pool_value_t *value); int pool_value_get_bool(const pool_value_t *value, uchar_t *bool); int pool_value_get_double(const pool_value_t *value, double *d); int pool_value_get_int64(const pool_value_t *value, int64_t *i64); int pool_value_get_string(const pool_value_t *value, const char **strp); int pool_value_get_uint64(const pool_value_t *value, uint64_t *ui64); void pool_value_set_bool(const pool_value_t *value, uchar_t bool); void pool_value_set_double(const pool_value_t *value, double d); void pool_value_set_int64(const pool_value_t *value, int64_t i64); int pool_value_set_string(const pool_value_t *value, const char *strp); void pool_value_set_uint64(const pool_value_t *value, uint64_t ui64); const char *pool_value_get_name(const pool_value_t *value); int pool_value_set_name(const pool_value_t *value, const char *name);</pre>
DESCRIPTION	A pool_value_t is an opaque type representing the typed value portion of a pool property. For a list of the types supported by a pool_value_t, see pool_get_property(3POOL). The pool_value_alloc() function allocates and returns an opaque container for a pool property value. The pool_value_free() function must be called explicitly for allocated property values. The pool_value_get_bool(), pool_value_get_double(), pool_value_get_int64(), pool_value_get_string(), and pool_value_get_uint64() functions retrieve the value contained in the

pool_value_alloc(3POOL)

pool_value_t pointed to by *value* to the location pointed to by the second argument. If the type of the value does not match that expected by the function, an error value is returned. The string retrieved by pool_value_get_string() is freed by the library when the value is overwritten or pool_value_free() is called on the pool property value.

The pool_value_get_type() function returns the type of the data contained by a pool_value_t. If the value is unused then a type of POC_INVALID is returned.

The pool_value_set_bool(), pool_value_set_double(), pool_value_set_int64(), pool_value_set_string(), and pool_value_set_uint64() functions set the value and type of the property value to the provided values. The pool_value_set_string() function copies the string passed in and returns -1 if the memory allocation fails.

Property values can optionally have names. These names are used to describe properties as name=value pairs in the various query functions (see pool_query_resources(3POOL)). A copy of the string passed to pool_value_set_name() is made by the library, and the value returned by pool_value_get_name() is freed when the pool_value_t is deallocated or overwritten.

RETURN VALUES

Upon successful completion, pool_value_alloc() returns a pool property value with type initialized to PVC_INVALID. Otherwise, NULL is returned and pool_error() returns the pool-specific error value.

Upon successful completion, pool_value_get_type() returns the type contained in the property value passed in as an argument. Otherwise, POC_INVALID is returned and pool_error() returns the pool-specific error value.

Upon successful completion, pool_value_get_bool(), pool_value_get_double(), pool_value_get_int64(), pool_value_get_string(), and pool_value_get_uint64() return 0. Otherwise -1 is returned and pool_error(3POOL) returns the pool-specific error value.

Upon successful completion, pool_value_set_string() and pool_value_set_name() return 0. If the memory allocation failed, -1 is returned and pool_error() returns the pool-specific error value.

ERRORS

The pool_value_alloc() function will fail if:

POE_SYSTEM	A system error has occurred. Check the system error code for more details.

The pool_value_get_bool(), pool_value_get_double(), pool_value_get_int64(), pool_value_get_string(), and pool_value_get_uint64() functions will fail if:

POE_BADPARAM	The supplied <i>value</i> does not match the type of the requested operation.

pool_value_alloc(3POOL)

The pool_value_set_string() function will fail if:

POE_SYSTEM A system error has occurred. Check the system error code for more details.

The pool_value_set_name() function will fail if:

POE_SYSTEM A system error has occurred. Check the system error code for more details.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
CSI	Enabled
Interface Stability	Unstable
MT-Level	Unsafe

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

pool_walk_components(3POOL)

NAME	pool_walk_components, pool_walk_pools, pool_walk_resources – walk objects within resource pool configurations						
SYNOPSIS	<pre>cc [flag...] file... -lpool [library...] #include <pool.h> int pool_walk_components(pool_conf_t *conf, pool_resource_t *resource, void *arg, int (*callback)(pool_conf_t *, pool_component_t *, void *)); int pool_walk_pools(pool_conf_t *conf, void *arg, int (*callback)(pool_conf_t *, pool_component_t *, void *)); int pool_walk_resources(pool_conf_t *conf, pool_t *pool, void *arg, int (*callback)(pool_conf_t *, pool_component_t *, void *));</pre>						
DESCRIPTION	The walker functions provided with libpool(3POOL) visit each associated entity of the given type, and call the caller-provided <i>callback</i> function with a user-provided additional opaque argument. There is no implied order of visiting nodes in the walk. If the <i>callback</i> function returns a non-zero value at any of the nodes, the walk is terminated, and an error value of -1 returned. The <i>conf</i> argument for each function refers to the target configuration to which the operation applies. The <code>pool_walk_components()</code> function invokes <i>callback</i> on all components contained in the resource. The <code>pool_walk_pools()</code> function invokes <i>callback</i> on all pools defined in the configuration. The <code>pool_walk_resources()</code> function invokes <i>callback</i> function on all resources associated with <i>pool</i>.						
RETURN VALUES	Upon successful completion of the walk, these functions return 0. Otherwise -1 is returned and <code>pool_error(3POOL)</code> returns the pool-specific error value.						
ERRORS	These functions will fail if: <table><tr><td>POE_BADPARAM</td><td>The supplied configuration's status is not POF_VALID.</td></tr><tr><td>POE_INVALID_CONF</td><td>The configuration is invalid.</td></tr><tr><td>POE_SYSTEM</td><td>A system error has occurred. Check the system error code for more details.</td></tr></table>	POE_BADPARAM	The supplied configuration's status is not POF_VALID.	POE_INVALID_CONF	The configuration is invalid.	POE_SYSTEM	A system error has occurred. Check the system error code for more details.
POE_BADPARAM	The supplied configuration's status is not POF_VALID.						
POE_INVALID_CONF	The configuration is invalid.						
POE_SYSTEM	A system error has occurred. Check the system error code for more details.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>CSI</td><td>Enabled</td></tr><tr><td>Interface Stability</td><td>Unstable</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	CSI	Enabled	Interface Stability	Unstable
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
CSI	Enabled						
Interface Stability	Unstable						

pool_walk_components(3POOL)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)

pow(3M)

NAME	pow – power function								
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double pow(double x, double y);</pre>								
DESCRIPTION	The <code>pow()</code> function computes the value of x raised to the power y , x^y . If x is negative, y must be an integer value.								
RETURN VALUES	Upon successful completion, <code>pow()</code> returns the value of x raised to the power y. If x is 0 and y is 0, 1.0 is returned. If y is NaN, or y is non-zero and x is NaN, NaN is returned. If y is 0.0 and x is NaN, NaN is returned. If x is 0.0 and y is negative, -HUGE_VAL is returned and <code>errno</code> may be set to <code>EDOM</code> or <code>ERANGE</code>. If the correct value would cause overflow, ±HUGE_VAL is returned, and <code>errno</code> is set to <code>ERANGE</code>. If the correct value would cause underflow to 0, 0 is returned and <code>errno</code> may be set to <code>ERANGE</code>. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.								
ERRORS	The <code>pow()</code> function will fail if: <table><tr><td>EDOM</td><td>The value of x is negative and y is non-integral.</td></tr><tr><td>ERANGE</td><td>The value to be returned would have caused overflow.</td></tr></table> The <code>pow()</code> function may fail if: <table><tr><td>EDOM</td><td>The value of x is 0.0 and y is negative.</td></tr><tr><td>ERANGE</td><td>The correct value would cause underflow.</td></tr></table>	EDOM	The value of x is negative and y is non-integral.	ERANGE	The value to be returned would have caused overflow.	EDOM	The value of x is 0.0 and y is negative.	ERANGE	The correct value would cause underflow.
EDOM	The value of x is negative and y is non-integral.								
ERANGE	The value to be returned would have caused overflow.								
EDOM	The value of x is 0.0 and y is negative.								
ERANGE	The correct value would cause underflow.								
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>pow()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	MT-Safe								
SEE ALSO	<code>exp(3M)</code> , <code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>								

printDmiAttributeValues(3DMI)

NAME	printDmiAttributeValues – print data in input DmiAttributeValues list				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <dmi/util.hh> void printDmiAttributeValues(DmiAttributeValues_t *<i>values</i>);</pre>				
DESCRIPTION	The <code>printDmiAttributeValues()</code> function prints the data in the input DmiAttributeValues list. The function prints "unknown data" for those <i>values</i> that contain invalid data.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-level	MT-Safe				
SEE ALSO	<code>libdmi(3LIB)</code> , <code>attributes(5)</code>				

printDmiDataUnion(3DMI)

NAME printDmiDataUnion – print data in input data union

SYNOPSIS

```
cc [ flag ... ] file ... -ldmi -lnsl -lrwtool [ library ... ]  
#include <dmi/util.hh>  
  
void printDmiDataUnion(DmiDataUnion_t *data);
```

DESCRIPTION The printDmiDataUnion() function prints the data in the input data union. The output depends on the type of DMI data in the union.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	MT-Safe

SEE ALSO libdmi(3LIB), attributes(5)

NAME printDmiString – print a DmiString

SYNOPSIS `cc [ flag ... ] file ... -ldmi -lnsl -lrwtool [ library ... ]`
`#include <dmi/util.hh>`

`void printDmiString(DmiString_t *dstr);`

DESCRIPTION The printDmiString() function prints a DmiString.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-level	MT-Safe

SEE ALSO newDmiString(3DMI), libdmi(3LIB), attributes(5)

project_walk(3PROJECT)

NAME	project_walk – visit active project IDs on current system		
SYNOPSIS	<pre>cc [flag...] file... -lproject [library...] #include <project.h> int project_walk(int (*callback)(const projid_t project, void *walk_data), void *init_data);</pre>		
DESCRIPTION	The <code>project_walk()</code> function provides a mechanism for the application author to examine all active projects on the current system. The <i>callback</i> function provided by the application is given the ID of an active project at each invocation and can use the <i>walk_data</i> to record its own state. The callback function should return non-zero if it encounters an error condition or attempts to terminate the walk prematurely; otherwise the callback function should return 0.		
RETURN VALUES	Upon successful completion, <code>project_walk()</code> returns 0. It returns -1 if the <i>callback</i> function returned a non-zero value or if the walk encountered an error, in which case <code>errno</code> is set to indicate the error.		
ERRORS	The <code>project_walk()</code> function will fail if: <table><tr><td>ENOMEM</td><td>There is insufficient memory available to set up the initial data for the walk.</td></tr></table> Other returned error values are presumably caused by the <i>callback</i> function.	ENOMEM	There is insufficient memory available to set up the initial data for the walk.
ENOMEM	There is insufficient memory available to set up the initial data for the walk.		
EXAMPLES	EXAMPLE 1 Count the number of projects available on the system. The following example counts the number of projects available on the system. <pre>#include <sys/types.h> #include <project.h> #include <stdio.h> typedef struct wdata { uint_t count; } wdata_t; wdata_t total_count; int simple_callback(const projid_t p, void *pvt) { wdata_t *w = (wdata_t *)pvt; w->count++; return (0); } ... total_count.count = 0; errno = 0; if (n=project_walk(simple_callback, &total_count)) >= 0) (void) printf("count = %u\n", total_count.count);</pre>		

project_walk(3PROJECT)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO getprojid(2), settaskid(2), attributes(5)

ptree_add_node(3PICLTREE)

NAME	ptree_add_node, ptree_delete_node – add or delete node to or from tree												
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_add_node(picl_nodehdl_t <i>parh</i>, picl_nodehdl_t <i>chdh</i>) ; int ptree_delete_node(ptree_delete_node <i>nodeh</i>) ;</pre>												
DESCRIPTION	The <code>ptree_add_node()</code> function adds the node specified by handle <i>chdh</i> as a child node to the node specified by the handle <i>parh</i>. <code>PICL_CANTPARENT</code> is if the child node already has a parent. The <code>ptree_delete_node()</code> function deletes the node specified by handle <i>nodeh</i> and all its descendant nodes from the tree.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed.												
ERRORS	<table><tr><td><code>PICL_NOTNODE</code></td><td>Node a node</td></tr><tr><td><code>PICL_CANTPARENT</code></td><td>Already has a parent</td></tr><tr><td><code>PICL_TREEBUSY</code></td><td>PICL tree is busy</td></tr><tr><td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle</td></tr><tr><td><code>PICL_STALEHANDLE</code></td><td>Stale handle</td></tr><tr><td><code>PICL_FAILURE</code></td><td>General system failure</td></tr></table>	<code>PICL_NOTNODE</code>	Node a node	<code>PICL_CANTPARENT</code>	Already has a parent	<code>PICL_TREEBUSY</code>	PICL tree is busy	<code>PICL_INVALIDHANDLE</code>	Invalid handle	<code>PICL_STALEHANDLE</code>	Stale handle	<code>PICL_FAILURE</code>	General system failure
<code>PICL_NOTNODE</code>	Node a node												
<code>PICL_CANTPARENT</code>	Already has a parent												
<code>PICL_TREEBUSY</code>	PICL tree is busy												
<code>PICL_INVALIDHANDLE</code>	Invalid handle												
<code>PICL_STALEHANDLE</code>	Stale handle												
<code>PICL_FAILURE</code>	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	<code>attributes(5)</code>												

ptree_add_prop(3PICLTREE)

NAME	ptree_add_prop, ptree_delete_prop – add or delete a property												
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_add_prop(picl_nodehdl_t nodeh, picl_prophdl_t proph); int ptree_delete_prop(picl_prophdl_t proph);</pre>												
DESCRIPTION	The ptree_add_prop() function adds the property specified by the handle <i>proph</i> to the list of properties of the node specified by handle <i>nodeh</i>. The ptree_delete_prop() function deletes the property from the property list of the node. For a table property, the entire table is deleted.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.												
ERRORS	<table><tr><td>PICL_NOTTABLE</td><td>Not a table</td></tr><tr><td>PICL_NOTPROP</td><td>Not a property</td></tr><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_PROPEXISTS</td><td>Property already exists</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTTABLE	Not a table	PICL_NOTPROP	Not a property	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_PROPEXISTS	Property already exists	PICL_FAILURE	General system failure
PICL_NOTTABLE	Not a table												
PICL_NOTPROP	Not a property												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_PROPEXISTS	Property already exists												
PICL_FAILURE	General system failure												
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	ptree_create_prop(3PICLTREE), attributes(5)												

ptree_create_and_add_node(3PICLTREE)

NAME	ptree_create_and_add_node – create and add node to tree and return node handle																
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_create_and_add_node(picl_nodehdl_t <i>parh</i>, const char *<i>name</i>, const char *<i>classname</i>, picl_nodehdl_t *<i>nodeh</i>);</pre>																
DESCRIPTION	The <code>ptree_create_and_add_node()</code> function creates a node with the name and PICL class specified by <i>name</i> and <i>classname</i> respectively. It then adds the node as a child to the node specified by <i>parh</i> . The handle of the new node is returned in <i>nodeh</i> .																
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.																
ERRORS	<table><tr><td>PICL_INVALIDARG</td><td>Invalid argument</td></tr><tr><td>PICL_VALUETOOBIG</td><td>Value exceeds maximum size</td></tr><tr><td>PICL_NOTSUPPORTED</td><td>Property version not supported</td></tr><tr><td>PICL_CANTDESTROY</td><td>Attempting to destroy before delete</td></tr><tr><td>PICL_NOTNODE</td><td>Not a node</td></tr><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_INVALIDARG	Invalid argument	PICL_VALUETOOBIG	Value exceeds maximum size	PICL_NOTSUPPORTED	Property version not supported	PICL_CANTDESTROY	Attempting to destroy before delete	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_INVALIDARG	Invalid argument																
PICL_VALUETOOBIG	Value exceeds maximum size																
PICL_NOTSUPPORTED	Property version not supported																
PICL_CANTDESTROY	Attempting to destroy before delete																
PICL_NOTNODE	Not a node																
PICL_INVALIDHANDLE	Invalid handle																
PICL_STALEHANDLE	Stale handle																
PICL_FAILURE	General system failure																
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe										
ATTRIBUTE TYPE	ATTRIBUTE VALUE																
Interface Stability	Evolving																
MT-Level	MT-Safe																
SEE ALSO	<code>ptree_create_node(3PICLTREE)</code> , <code>ptree_add_node(3PICLTREE)</code> , <code>attributes(5)</code>																

	ptree_create_and_add_prop(3PICLTREE)							
NAME	ptree_create_and_add_prop – create and add property to node and return property handle							
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_create_and_add_prop(picl_nodehdl_t <i>nodeh</i>, ptree_propinfo_t *<i>infop</i>, void *<i>vbuf</i>, picl_prophdl_t *<i>proph</i>) ;</pre>							
DESCRIPTION	The ptree_create_and_add_prop() function creates a property using the the property information specified in <i>infop</i> and the value buffer <i>vbuf</i> and adds the property to the node specified by <i>nodeh</i> . If <i>proph</i> is not NULL, the handle of the property added to the node is returned in <i>proph</i> .							
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.							
	PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized.							
	PICL_INVALIDHANDLE is returned if the specified handle never existed.							
ERRORS	PICL_NOTSUPPORTED	Property version not supported						
	PICL_VALUETOOBIG	Value exceeds maximum size						
	PICL_NOTPROP	Not a property						
	PICL_NOTTABLE	Not a table						
	PICL_PROPEXISTS	Property already exists						
	PICL_RESERVEDNAME	Property name is reserved						
	PICL_INVREFERENCE	Invalid reference property value						
	PICL_INVALIDHANDLE	Invalid handle						
	PICL_STALEHANDLE	Stale handle						
	PICL_FAILURE	General system failure						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Interface Stability	Evolving							
MT-Level	MT-Safe							
SEE ALSO	ptree_create_prop(3PICLTREE), ptree_add_prop(3PICLTREE), attributes(5)							

ptree_create_node(3PICLTREE)

NAME	ptree_create_node, ptree_destroy_node – create or destroy a node																		
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_create_node(char *name, char *cname, picl_nodehdl_t *nodeh) ; int ptree_destroy_node(picl_nodehdl_t nodeh) ;</pre>																		
DESCRIPTION	The <code>ptree_create_node()</code> function creates a node and sets the "name" property value to the string specified in <i>name</i> and the "class" property value to the string specified in <i>cname</i>. The handle of the new node is copied into the location given by <i>nodeh</i>. The <code>ptree_destroy_node()</code> function destroys the node specified by <i>nodeh</i> and frees up any allocated space. The node to be destroyed must have been previously deleted by <code>ptree_delete_node</code> (see <code>ptree_add_node(3PICLTREE)</code>). Otherwise, <code>PICL_CANTDESTROY</code> is returned.																		
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed.																		
ERRORS	<table> <tr> <td><code>PICL_INVALIDARG</code></td><td>Invalid argument</td></tr> <tr> <td><code>PICL_VALUETOOBIG</code></td><td>Value exceeds maximum size</td></tr> <tr> <td><code>PICL_NOTSUPPORTED</code></td><td>Property version not supported</td></tr> <tr> <td><code>PICL_CANTDESTROY</code></td><td>Attempting to destroy before delete</td></tr> <tr> <td><code>PICL_TREEBUSY</code></td><td>PICL tree is busy</td></tr> <tr> <td><code>PICL_NOTNODE</code></td><td>Not a node</td></tr> <tr> <td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle</td></tr> <tr> <td><code>PICL_STALEHANDLE</code></td><td>Stale handle</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> </table>	<code>PICL_INVALIDARG</code>	Invalid argument	<code>PICL_VALUETOOBIG</code>	Value exceeds maximum size	<code>PICL_NOTSUPPORTED</code>	Property version not supported	<code>PICL_CANTDESTROY</code>	Attempting to destroy before delete	<code>PICL_TREEBUSY</code>	PICL tree is busy	<code>PICL_NOTNODE</code>	Not a node	<code>PICL_INVALIDHANDLE</code>	Invalid handle	<code>PICL_STALEHANDLE</code>	Stale handle	<code>PICL_FAILURE</code>	General system failure
<code>PICL_INVALIDARG</code>	Invalid argument																		
<code>PICL_VALUETOOBIG</code>	Value exceeds maximum size																		
<code>PICL_NOTSUPPORTED</code>	Property version not supported																		
<code>PICL_CANTDESTROY</code>	Attempting to destroy before delete																		
<code>PICL_TREEBUSY</code>	PICL tree is busy																		
<code>PICL_NOTNODE</code>	Not a node																		
<code>PICL_INVALIDHANDLE</code>	Invalid handle																		
<code>PICL_STALEHANDLE</code>	Stale handle																		
<code>PICL_FAILURE</code>	General system failure																		
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe												
ATTRIBUTE TYPE	ATTRIBUTE VALUE																		
Interface Stability	Evolving																		
MT-Level	MT-Safe																		

`ptree_create_node(3PICLTREE)`

SEE ALSO `ptree_add_node(3PICLTREE), attributes(5)`

ptree_create_prop(3PICLTREE)

NAME	ptree_create_prop, ptree_destroy_prop – create or destroy a property																		
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_create_prop(ptree_propinfo_t *<i>pinfo</i>, void *<i>valbuf</i>, picl_prophdl_t *<i>proph</i>) ; int ptree_destroy_prop(picl_prophdl_t <i>proph</i>) ;</pre>																		
DESCRIPTION	The <code>ptree_create_prop()</code> function creates a property using the information specified in <i>pinfo</i>, which includes the name, type, access mode, and size of the property, as well as the read access function for a volatile property. The value of the property is specified in the buffer <i>valbuf</i>, which may be NULL for volatile properties. The handle of the property created is copied into the location given by <i>proph</i>. See <code>libpicltree(3PICLTREE)</code> for more information on the structure of <code>ptree_propinfo_t</code> structure. The <code>ptree_destroy_prop()</code> function destroys the property specified by the handle <i>proph</i>. For a table property, the entire table is destroyed. The property to be destroyed must have been previously deleted.																		
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.																		
ERRORS	<table> <tr> <td>PICL_NOTSUPPORTED</td><td>Property version not supported</td></tr> <tr> <td>PICL_VALUETOOBIG</td><td>Value exceeds maximum size</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_CANTDESTROY</td><td>Attempting to destroy before delete</td></tr> <tr> <td>PICL_RESERVEDNAME</td><td>Property name is reserved</td></tr> <tr> <td>PICL_INVREFERENCE</td><td>Invalid reference property value</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_NOTSUPPORTED	Property version not supported	PICL_VALUETOOBIG	Value exceeds maximum size	PICL_NOTPROP	Not a property	PICL_CANTDESTROY	Attempting to destroy before delete	PICL_RESERVEDNAME	Property name is reserved	PICL_INVREFERENCE	Invalid reference property value	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_NOTSUPPORTED	Property version not supported																		
PICL_VALUETOOBIG	Value exceeds maximum size																		
PICL_NOTPROP	Not a property																		
PICL_CANTDESTROY	Attempting to destroy before delete																		
PICL_RESERVEDNAME	Property name is reserved																		
PICL_INVREFERENCE	Invalid reference property value																		
PICL_INVALIDHANDLE	Invalid handle																		
PICL_STALEHANDLE	Stale handle																		
PICL_FAILURE	General system failure																		
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving														
ATTRIBUTE TYPE	ATTRIBUTE VALUE																		
Interface Stability	Evolving																		

ptree_create_prop(3PICLTREE)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO libpicltree(3PICLTREE), ptree_add_prop(3PICLTREE), attributes(5)

ptree_create_table(3PICLTREE)

NAME	ptree_create_table, ptree_add_row_to_table – create a table object												
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_create_table(picl_prophdl_t *tbl_hdl) ; int ptree_add_row_to_table(picl_prophdl_t tbl_hdl, int nprops, picl_prophdl_t *proph) ;</pre>												
DESCRIPTION	The <code>ptree_create_table()</code> function creates a table object and returns the handle of the table in <i>tbl_hdl</i>. The <code>ptree_add_row_to_table()</code> function adds a row of properties to the table specified by <i>tbl_hdl</i>. The handles of the properties of the row are specified in the <i>proph</i> array and <i>nprops</i> specifies the number of handles in the array. The number of columns in the table is determined from the first row added to the table. If extra column values are specified in subsequent rows, they are ignored. The row is appended to the end of the table.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.												
ERRORS	<table> <tr> <td>PICL_INVALIDARG</td><td>Invalid argument</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_NOTTABLE</td><td>Not a table</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_INVALIDARG	Invalid argument	PICL_NOTPROP	Not a property	PICL_NOTTABLE	Not a table	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_INVALIDARG	Invalid argument												
PICL_NOTPROP	Not a property												
PICL_NOTTABLE	Not a table												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_FAILURE	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	<code>attributes(5)</code>												

ptree_find_node(3PICLTREE)

NAME	ptree_find_node – find node with given property and value										
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_find_node(picl_nodehdl_t <i>rooth</i>, char *<i>pname</i>, picl_prop_type_t <i>ptype</i>, void *<i>pval</i>, size_t <i>valsize</i>, picl_nodehdl_t *<i>retnodeh</i>) ;</pre>										
DESCRIPTION	The <code>ptree_find_node()</code> function visits the nodes in the subtree under the node specified by <i>rooth</i>. The handle of the node that has the property whose name, type, and value matches the name, type, and value specified in <i>pname</i>, <i>ptype</i>, and <i>pval</i> respectively, is returned in the location given by <i>retnodeh</i>. The argument <i>valsize</i> gives the size of the value in <i>pval</i>. The first <i>valsize</i> number of bytes of the property value is compared with <i>pval</i>.										
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_NODENOTFOUND is returned if there is no node that matches the property criteria can be found.										
ERRORS	<table> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_PROPNOTFOUND	Property not found	PICL_FAILURE	General system failure
PICL_NOTNODE	Not a node										
PICL_INVALIDHANDLE	Invalid handle										
PICL_STALEHANDLE	Stale handle										
PICL_PROPNOTFOUND	Property not found										
PICL_FAILURE	General system failure										
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Interface Stability	Evolving										
MT-Level	MT-Safe										
SEE ALSO	<code>ptree_get_prop_by_name(3PICLTREE)</code>, <code>ptree_get_propinfo(3PICLTREE)</code>, <code>ptree_get_propval(3PICLTREE)</code>, <code>ptree_get_propval_by_name(3PICLTREE)</code>, <code>attributes(5)</code>										

ptree_get_first_prop(3PICLTREE)

NAME	ptree_get_first_prop, ptree_get_next_prop – get a property handle of the node												
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_get_first_prop(picl_nodehdl_t <i>nodeh</i>, picl_prophdl_t *<i>proph</i>) ; int ptree_get_next_prop(picl_prophdl_t <i>proph</i>, picl_prophdl_t *<i>nextproph</i>) ;</pre>												
DESCRIPTION	The <code>ptree_get_first_prop()</code> function gets the handle of the first property of the node specified by <i>nodeh</i> and copies it into the location specified by <i>proph</i>. The <code>ptree_get_next_prop()</code> function gets the handle of the next property after the one specified by <i>proph</i> from the list of properties of the node and copies it into the location specified by <i>nextproph</i>.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.												
ERRORS	<table> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_ENDOFLIST</td><td>End of list</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_NOTPROP	Not a property	PICL_NOTNODE	Not a node	PICL_ENDOFLIST	End of list	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_NOTPROP	Not a property												
PICL_NOTNODE	Not a node												
PICL_ENDOFLIST	End of list												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_FAILURE	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	ptree_get_prop_by_name(3PICLTREE), attributes(5)												

	ptree_get_next_by_row(3PICLTREE)										
NAME	ptree_get_next_by_row, ptree_get_next_by_col – access a table property										
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_get_next_by_row(picl_prophdl_t proph, picl_prophdl_t *rowh) ; int ptree_get_next_by_col(picl_prophdl_t proph, picl_prophdl_t *colh) ;</pre>										
DESCRIPTION	The <code>ptree_get_next_by_row()</code> function copies the handle of the property that is in the next column of the table and on the same row as the property <i>proph</i>. The handle is copied into the location given by <i>rowh</i>. The <code>ptree_get_next_by_col()</code> function copies the handle of the property that is in the next row of the table and on the same column as the property <i>proph</i>. The handle is copied into the location given by <i>colh</i>. If there are no more rows or columns, this function returns the value <code>PICL_ENDOFLIST</code>.										
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_STALEHANDLE</code> is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. <code>PICL_INVALIDHANDLE</code> is returned if the specified handle never existed.										
ERRORS	<table> <tr> <td><code>PICL_NOTTABLE</code></td><td>Not a table</td></tr> <tr> <td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle</td></tr> <tr> <td><code>PICL_STALEHANDLE</code></td><td>Stale handle</td></tr> <tr> <td><code>PICL_ENDOFLIST</code></td><td>End of list</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> </table>	<code>PICL_NOTTABLE</code>	Not a table	<code>PICL_INVALIDHANDLE</code>	Invalid handle	<code>PICL_STALEHANDLE</code>	Stale handle	<code>PICL_ENDOFLIST</code>	End of list	<code>PICL_FAILURE</code>	General system failure
<code>PICL_NOTTABLE</code>	Not a table										
<code>PICL_INVALIDHANDLE</code>	Invalid handle										
<code>PICL_STALEHANDLE</code>	Stale handle										
<code>PICL_ENDOFLIST</code>	End of list										
<code>PICL_FAILURE</code>	General system failure										
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Interface Stability	Evolving										
MT-Level	MT-Safe										
SEE ALSO	<code>ptree_create_table(3PICLTREE)</code> , <code>attributes(5)</code>										

ptree_get_node_by_path(3PICLTREE)

NAME	ptree_get_node_by_path – get handle of node specified by PICL tree path						
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_get_node_by_path(const char *ptreepath, picl_nodehdl_t *nodeh) ;</pre>						
DESCRIPTION	The <code>ptree_get_node_by_path()</code> function copies the handle of the node in the PICL tree specified by the path given in <i>ptreepath</i> into the location <i>nodeh</i>. The syntax of a PICL tree path is: <pre>[def_propname:] / [def_propval [match_cond] ...]</pre> where <i>def_propname</i> prefix is a shorthand notation to specify the name of the property whose value is specified in <i>def_propval</i>, and the <i>match_cond</i> expression specifies the matching criteria for that node in the form of one or more pairs of property names and values such as <pre>[@address] [?prop_name [=prop_val] ...]</pre> where '@' is a shorthand notation to refer to the device address, which is followed by the device address value address. The address value is matched with the value of the property "bus-addr" if it exists. If no "bus-addr" property exists, then it is matched with the value of the property "UnitAddress". Use the '?' notation to limit explicitly the comparison to "bus-addr" or "UnitAddress" property. The expression following '?' specifies matching property name and value pairs, where <i>prop_name</i> gives the property name and <i>prop_val</i> gives the property value for non <code>PICL_PTYPE_VOID</code> properties. The values for properties of type <code>PICL_PTYPE_TABLE</code>, <code>PICL_PTYPE_BYTEARRAY</code>, and <code>PICL_PTYPE_REFERENCE</code> cannot be specified in the <i>match_cond</i> expression. A "_class" property value of "picl" may be used to match nodes of all PICL classes. All valid paths must start at the root node denoted by '/'. If no prefix is specified for the path, then the prefix defaults to the "name" property.						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. <code>PICL_NOTNODE</code> is returned if there is no node corresponding to the specified path.						
ERRORS	<table> <tr> <td><code>PICL_INVALIDARG</code></td><td>Invalid argument</td></tr> <tr> <td><code>PICL_NOTNODE</code></td><td>Not a node</td></tr> <tr> <td><code>PICL_FAILURE</code></td><td>General system failure</td></tr> </table>	<code>PICL_INVALIDARG</code>	Invalid argument	<code>PICL_NOTNODE</code>	Not a node	<code>PICL_FAILURE</code>	General system failure
<code>PICL_INVALIDARG</code>	Invalid argument						
<code>PICL_NOTNODE</code>	Not a node						
<code>PICL_FAILURE</code>	General system failure						

`ptree_get_node_by_path(3PICLTREE)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `ptree_get_propval_by_name(3PICLTREE)`, `attributes(5)`

ptree_get_prop_by_name(3PICLTREE)

NAME	ptree_get_prop_by_name – get a property handle by name												
SYNOPSIS	<pre>cc [flag] file... -lpicltree [library...] #include <picltree.h> int ptree_get_prop_by_name(picl_nodehdl_t nodeh, char *name, picl_prophdl_t *proph);</pre>												
DESCRIPTION	The <code>ptree_get_prop_by_name()</code> function gets the handle of the property, whose name is specified in <i>name</i> , of the node specified by the handle <i>nodeh</i> . The property handle is copied into the location specified by <i>proph</i> .												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_RESERVEDNAME is returned if the name specified is a PICL reserved name property. Reserved name properties do not have an associated property handle. Use <code>ptree_get_propval_by_name(3PICLTREE)</code> to get the value of a reserved property. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.												
ERRORS	<table><tr><td>PICL_NOTNODE</td><td>Not a node</td></tr><tr><td>PICL_RESERVEDNAME</td><td>Property name is reserved</td></tr><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTNODE	Not a node	PICL_RESERVEDNAME	Property name is reserved	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_PROPNOTFOUND	Property not found	PICL_FAILURE	General system failure
PICL_NOTNODE	Not a node												
PICL_RESERVEDNAME	Property name is reserved												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_PROPNOTFOUND	Property not found												
PICL_FAILURE	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	<code>ptree_get_first_prop(3PICLTREE)</code> , <code>ptree_get_propval_by_name(3PICLTREE)</code> , <code>attributes(5)</code>												

NAME	ptree_get_propinfo – get property information								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_get_propinfo(picl_prophdl_t <i>proph</i>, ptree_propinfo_t *<i>pi</i>) ;</pre>								
DESCRIPTION	The <code>ptree_get_propinfo()</code> function gets the information about the property specified by handle <i>proph</i> and copies it into the location specified by <i>pi</i> . See <code>libpicltree(3PICLTREE)</code> for more information about <code>ptree_propinfo_t</code> structure.								
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.								
ERRORS	<table><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_NOTPROP</td><td>Not a property</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_NOTPROP	Not a property	PICL_FAILURE	General system failure
PICL_INVALIDHANDLE	Invalid handle								
PICL_STALEHANDLE	Stale handle								
PICL_NOTPROP	Not a property								
PICL_FAILURE	General system failure								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Evolving								
MT-Level	MT-Safe								
SEE ALSO	<code>libpicltree(3PICLTREE)</code> , <code>ptree_create_prop(3PICLTREE)</code> , <code>attributes(5)</code>								

ptree_get_propinfo_by_name(3PICLTREE)

NAME	ptree_get_propinfo_by_name – get property information and handle of named property												
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_get_propinfo_by_name(picl_nodehdl_t <i>nodeh</i>, const char *<i>pname</i>, ptree_propinfo_t *<i>pinfo</i>, picl_prophdl_t *<i>proph</i>);</pre>												
DESCRIPTION	The <code>ptree_get_propinfo_by_name()</code> function copies the property information of the property specified by <i>pname</i> in the node <i>nodeh</i> into the location given by <i>pinfo</i> . The handle of the property is returned in the location <i>proph</i> .												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.												
ERRORS	<table><tr><td>PICL_NOTNODE</td><td>Not a node</td></tr><tr><td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr><tr><td>PICL_RESERVEDNAME</td><td>Reserved property name specified</td></tr><tr><td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr><tr><td>PICL_STALEHANDLE</td><td>Stale handle</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_NOTNODE	Not a node	PICL_PROPNOTFOUND	Property not found	PICL_RESERVEDNAME	Reserved property name specified	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_FAILURE	General system failure
PICL_NOTNODE	Not a node												
PICL_PROPNOTFOUND	Property not found												
PICL_RESERVEDNAME	Reserved property name specified												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_FAILURE	General system failure												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	<code>picl_get_propinfo(3PICLTREE)</code> , <code>picl_get_prop_by_name(3PICLTREE)</code> , <code>attributes(5)</code>												

	ptree_get_propval(3PICLTREE)														
NAME	ptree_get_propval, ptree_get_propval_by_name – get the value of a property														
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_get_propval(picl_prophdl_t proph, void *valbuf, size_t nbytes); int ptree_get_propval_by_name(picl_nodehdl_t nodeh, void *name, void *valbuf, size_t nbytes);</pre>														
DESCRIPTION	The <code>ptree_get_propval()</code> function gets the value of the property specified by the handle <i>proph</i> and copies it into the buffer specified by <i>valbuf</i>. The size of the buffer <i>valbuf</i> is specified in <i>nbytes</i>. The <code>ptree_get_propval_by_name()</code> function gets the value of the property, whose name is specified by <i>name</i>, from the node specified by handle <i>nodeh</i>. The value is copied into the buffer specified by <i>valbuf</i>. The size of the buffer is specified by <i>nbytes</i>. For volatile properties, the read access function provided by the plug-in publishing the property is invoked.														
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.														
ERRORS	<table> <tr> <td>PICL_VALUETOOBIG</td><td>Value too big</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_VALUETOOBIG	Value too big	PICL_NOTPROP	Not a property	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_PROPNOTFOUND	Property not found	PICL_FAILURE	General system failure
PICL_VALUETOOBIG	Value too big														
PICL_NOTPROP	Not a property														
PICL_NOTNODE	Not a node														
PICL_INVALIDHANDLE	Invalid handle														
PICL_STALEHANDLE	Stale handle														
PICL_PROPNOTFOUND	Property not found														
PICL_FAILURE	General system failure														
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe										
ATTRIBUTE TYPE	ATTRIBUTE VALUE														
MT-Level	MT-Safe														
SEE ALSO	<code>ptree_update_propval(3PICLTREE)</code> , <code>attributes(5)</code>														

ptree_get_root(3PICLTREE)

NAME ptree_get_root – get the root node handle

SYNOPSIS cc [*flag...*] *file...* -lpicltree [*library...*]
#include <picltree.h>

```
int ptree_get_root(picl_nodehdl_t *nodeh) ;
```

DESCRIPTION The ptree_get_root() function copies the handle of the root node of the PICL tree into the location specified by *nodeh*.

RETURN VALUES Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.

ERRORS PICL_INVALIDARG Invalid argument
PICL_FAILURE General system failure

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libpicltree(3PICLTREE), ptree_create_node(3PICLTREE), attributes(5)

ptree_init_propinfo(3PICLTREE)

NAME	ptree_init_propinfo – initialize ptree_propinfo_t structure						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lpicltree [<i>library...</i>] #include <picltree.h> int ptree_init_propinfo(ptree_propinfo_t *<i>infop</i>, int <i>version</i>, int <i>ptype</i>, int <i>pmode</i>, size_t <i>psize</i>, char *<i>pname</i>, int (*<i>readfn</i>)(ptree_rarg_t *, void *), int (*<i>writfn</i>)(ptree_warg_t *, const void *));</pre>						
DESCRIPTION	The <code>ptree_init_propinfo()</code> function initializes a <code>ptree_propinfo_t</code> property information structure given by location <i>infop</i> with the values provided by the arguments. The <i>version</i> argument specifies the version of the <code>ptree_propinfo_t</code> structure. <code>PTREE_PROPINFO_VERSION</code> gives the current version. The arguments <i>ptype</i>, <i>pmode</i>, <i>psize</i>, and <i>pname</i> specify the property's PICL type, access mode, size, and name. The maximum size of a property name is defined by <code>PICL_PROPNAMLEN_MAX</code>. The arguments <i>readfn</i> and <i>writfn</i> specify a volatile property's read and write access functions. For non-volatile properties, these are set to <code>NULL</code>.						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.						
ERRORS	<table><tr><td><code>PICL_INVALIDARG</code></td><td>Invalid argument</td></tr><tr><td><code>PICL_NOTSUPPORTED</code></td><td>Property version not supported</td></tr><tr><td><code>PICL_FAILURE</code></td><td>General system failure</td></tr></table>	<code>PICL_INVALIDARG</code>	Invalid argument	<code>PICL_NOTSUPPORTED</code>	Property version not supported	<code>PICL_FAILURE</code>	General system failure
<code>PICL_INVALIDARG</code>	Invalid argument						
<code>PICL_NOTSUPPORTED</code>	Property version not supported						
<code>PICL_FAILURE</code>	General system failure						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>ptree_get_propinfo(3PICLTREE)</code> , <code>attributes(5)</code>						

ptree_post_event(3PICLTREE)

NAME	ptree_post_event – post a PICL event						
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_post_event(const char *ename, const void *earg, size_t size, void (*completion_handler)(char *ename, void *earg, size_t size)) ;</pre>						
DESCRIPTION	The <code>ptree_post_event()</code> function posts the specified event and its arguments to the PICL framework. The argument <i>ename</i> specifies a pointer to a string containing the name of the PICL event. The arguments <i>earg</i> and <i>size</i> specify a pointer to a buffer containing the event arguments and size of that buffer, respectively. The argument <i>completion_handler</i> specifies the completion handler to be called after the event has been dispatched to all handlers. A NULL value for a completion handler indicates that no handler should be called. The PICL framework invokes the completion handler of an event with the <i>ename</i>, <i>earg</i>, and <i>size</i> arguments specified at the time of the posting of the event. PICL events are dispatched in the order in which they were posted. They are dispatched by executing the handlers registered for that event. The handlers are invoked in the order in which they were registered. New events will not begin execution until all previous events have finished execution. Specifically, an event posted from an event handler will not begin execution until the current event has finished execution. The caller may not reuse or reclaim the resources associated with the event name and arguments until the invocation of the completion handler. The completion handlers are normally used to reclaim any resources allocated for the posting of an event.						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error, the event is not posted, and the completion handler is not invoked.						
ERRORS	<table><tr><td>PICL_INVALIDARG</td><td>Invalid argument</td></tr><tr><td>PICL_FAILURE</td><td>General system failure</td></tr></table>	PICL_INVALIDARG	Invalid argument	PICL_FAILURE	General system failure		
PICL_INVALIDARG	Invalid argument						
PICL_FAILURE	General system failure						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>ptree_register_handler(3PICLTREE)</code> , <code>ptree_unregister_handler(3PICLTREE)</code> , <code>attributes(5)</code>						

NAME	ptree_register_handler – register a handler for the event						
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_register_handler(const char *ename, void (*evt_handler)(const char *ename, const void *earg, size_t size, void *cookie), void *cookie);</pre>						
DESCRIPTION	The <code>ptree_register_handler()</code> function registers an event handler for a PICL event. The argument <i>ename</i> specifies the name of the PICL event for which to register the handler. The argument <i>evt_handler</i> specifies the event handler function. The argument <i>cookie</i> is a pointer to caller-specific data to be passed as an argument to the event handler when it is invoked. The event handler function must be defined as <pre>void evt_handler(const char *ename, const void *earg, \ size_t size, void *cookie)</pre> where, <i>ename</i>, <i>earg</i>, <i>size</i>, and <i>cookie</i> are the arguments passed to the event handler when it is invoked. The argument <i>ename</i> is the PICL event name for which the handler is invoked. The arguments <i>earg</i> and <i>size</i> gives the pointer to the event argument buffer and its size, respectively. The argument <i>cookie</i> is the pointer to the caller specific data registered with the handler. The arguments <i>ename</i> and <i>earg</i> point to buffers that are transient and shall not be modified by the event handler or reused after the event handler finishes execution. The PICL framework invokes the event handlers in the order in which they were registered when dispatching an event. If the event handler execution order is required to be the same as the plug-in dependency order, then a plug-in should register its handlers from its init function. The handlers that do not have any ordering dependencies on other plug-in handlers can be registered at any time. The registered handler may be called at any time after this function is called.						
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error and the handler is not registered.						
ERRORS	<table> <tr> <td>PICL_INVALIDARG</td><td>Invalid argument</td></tr> <tr> <td>PICL_FAILURE</td><td>General system failure</td></tr> </table>	PICL_INVALIDARG	Invalid argument	PICL_FAILURE	General system failure		
PICL_INVALIDARG	Invalid argument						
PICL_FAILURE	General system failure						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>ptree_unregister_handler(3PICLTREE)</code> , <code>attributes(5)</code>						

ptree_unregister_handler(3PICLTREE)

NAME	ptree_unregister_handler – unregister the event handler for the event				
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> void ptree_register_handler(const char *ename, void (*evt_handler)(const char *ename, const void *earg, size_t size, void *cookie), void *cookie);</pre>				
DESCRIPTION	The <code>ptree_unregister_handler()</code> function unregisters the event handler for the specified event. The argument <i>ename</i> specifies the name of the PICL event for which to unregister the handler. The argument <i>evt_handler</i> specifies the event handler function. The argument <i>cookie</i> is the pointer to the caller-specific data given at the time of registration of the handler. If the handler being unregistered is currently executing, then this function will block until its completion. Because of this, locks acquired by the handlers should not be held across the call to <code>ptree_unregister_handler()</code> or a deadlock may result. The <code>ptree_unregister_handler()</code> function must not be invoked from the handler that is being unregistered.				
RETURN VALUES	This function does not return a value.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>ptree_register_handler(3PICLTREE)</code> , <code>attributes(5)</code>				

	ptree_update_propval(3PICLTREE)												
NAME	ptree_update_propval, ptree_update_propval_by_name – update a property value												
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_update_propval(picl_prophdl_t <i>proph</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>); int ptree_update_propval_by_name(picl_nodehdl_t <i>nodeh</i>, char *<i>name</i>, void *<i>valbuf</i>, size_t <i>nbytes</i>);</pre>												
DESCRIPTION	The <code>ptree_update_propval()</code> function updates the value of the property specified by <i>proph</i> with the value specified in the buffer <i>valbuf</i>. The size of the buffer <i>valbuf</i> is specified in <i>nbytes</i>. The <code>ptree_update_propval_by_name()</code> function updates the value of the property, whose name is specified by <i>name</i>, of the node specified by handle <i>nodeh</i>. The new value is specified in the buffer <i>valbuf</i>, whose size is specified in <i>nbytes</i>. For volatile properties, the write access function provided by the plug-in publishing the property is invoked.												
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error. PICL_STALEHANDLE is returned if the handle is no longer valid. This occurs if the PICL tree was refreshed or reinitialized. PICL_INVALIDHANDLE is returned if the specified handle never existed.												
ERRORS	<table> <tr> <td>PICL_VALUETOOBIG</td><td>Value too big</td></tr> <tr> <td>PICL_NOTPROP</td><td>Not a property</td></tr> <tr> <td>PICL_NOTNODE</td><td>Not a node</td></tr> <tr> <td>PICL_INVALIDHANDLE</td><td>Invalid handle</td></tr> <tr> <td>PICL_STALEHANDLE</td><td>Stale handle</td></tr> <tr> <td>PICL_PROPNOTFOUND</td><td>Property not found</td></tr> </table>	PICL_VALUETOOBIG	Value too big	PICL_NOTPROP	Not a property	PICL_NOTNODE	Not a node	PICL_INVALIDHANDLE	Invalid handle	PICL_STALEHANDLE	Stale handle	PICL_PROPNOTFOUND	Property not found
PICL_VALUETOOBIG	Value too big												
PICL_NOTPROP	Not a property												
PICL_NOTNODE	Not a node												
PICL_INVALIDHANDLE	Invalid handle												
PICL_STALEHANDLE	Stale handle												
PICL_PROPNOTFOUND	Property not found												
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Interface Stability	Evolving												
MT-Level	MT-Safe												
SEE ALSO	<code>ptree_get_propval(3PICLTREE)</code> , <code>attributes(5)</code>												

ptree_walk_tree_by_class(3PICLTREE)

NAME	ptree_walk_tree_by_class – walk subtree by class								
SYNOPSIS	<pre>cc [flag...] file... -lpicltree [library...] #include <picltree.h> int ptree_walk_tree_by_class(picl_nodehdl_t rooth, const char *classname, void *c_args, int (*callback)(picl_nodehdl_t nodeh, void *c_args));</pre>								
DESCRIPTION	The <code>ptree_walk_tree_by_class()</code> function visits all the nodes of the subtree under the node specified by <i>rooth</i>. The PICL class name of the visited node is compared with the class name specified by <i>classname</i>. If the class names match, the callback function specified by <i>callback</i> is called with the matching node handle and the argument provided in <i>c_args</i>. If the class name specified in <i>classname</i> is NULL, then the callback function is invoked for all the nodes. The return value from the callback function is used to determine whether to continue or terminate the tree walk. The callback function returns <code>PICL_WALK_CONTINUE</code> or <code>PICL_WALK_TERMINATE</code> to continue or terminate the tree walk.								
RETURN VALUES	Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.								
ERRORS	<table><tr><td><code>PICL_NOTNODE</code></td><td>Not a node</td></tr><tr><td><code>PICL_INVALIDHANDLE</code></td><td>Invalid handle specified</td></tr><tr><td><code>PICL_STALEHANDLE</code></td><td>Stale handle specified</td></tr><tr><td><code>PICL_FAILURE</code></td><td>General system failure</td></tr></table>	<code>PICL_NOTNODE</code>	Not a node	<code>PICL_INVALIDHANDLE</code>	Invalid handle specified	<code>PICL_STALEHANDLE</code>	Stale handle specified	<code>PICL_FAILURE</code>	General system failure
<code>PICL_NOTNODE</code>	Not a node								
<code>PICL_INVALIDHANDLE</code>	Invalid handle specified								
<code>PICL_STALEHANDLE</code>	Stale handle specified								
<code>PICL_FAILURE</code>	General system failure								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Interface Stability	Evolving								
MT-Level	MT-Safe								
SEE ALSO	<code>ptree_get_propval_by_name(3PICLTREE)</code> , <code>attributes(5)</code>								

NAME	read_vtoc, write_vtoc – read and write a disk's VTOC				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ladm [<i>library</i> ...] #include <sys/vtoc.h> int read_vtoc(int <i>fd</i>, struct vtoc *<i>vtoc</i>); int write_vtoc(int <i>fd</i>, struct vtoc *<i>vtoc</i>);</pre>				
DESCRIPTION	The <code>read_vtoc()</code> function returns the VTOC (volume table of contents) structure that is stored on the disk associated with the open file descriptor <i>fd</i>. The <code>write_vtoc()</code> function stores the VTOC structure on the disk associated with the open file descriptor <i>fd</i>. The <i>fd</i> argument refers to any slice on a raw disk.				
RETURN VALUES	Upon successful completion, <code>read_vtoc()</code> returns a positive integer indicating the slice index associated with the open file descriptor. Otherwise, it returns a negative integer indicating one of the following errors: VT_EIO An I/O error occurred. VT_ENOTSUP This operation is not supported on this disk. VT_ERROR An unknown error occurred. Upon successful completion, <code>write_vtoc()</code> returns 0. Otherwise, it returns a negative integer indicating one of the following errors: VT_EINVAL The VTOC contains an incorrect field. VT_EIO An I/O error occurred. VT_ENOTSUP This operation is not supported on this disk. VT_ERROR An unknown error occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>fmthard(1M)</code> , <code>format(1M)</code> , <code>prtvto(1M)</code> , <code>ioctl(2)</code> , <code>efi_alloc_and_init(3EXT)</code> , <code>attributes(5)</code> , <code>dkio(7I)</code>				
BUGS	The <code>write_vtoc()</code> function cannot write a VTOC on an unlabeled disk. Use <code>format(1M)</code> for this purpose.				

reg_ci_callback(3DMI)

NAME	reg_ci_callback – provide a component instrumentation with a transient program number				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -ldmici [<i>library</i> ...] #include <dmi/ci_callback_svc.hh> u_long reg_ci_callback() ;</pre>				
DESCRIPTION	The <code>reg_ci_callback()</code> function provides a component instrumentation with a transient program number. The instrumentation uses this number to register its RPC service provider. The <code>prognum</code> member of the <code>DmiRegisterInfo</code> structure is populated with the return value of this function				
RETURN VALUES	Upon successful completion, the <code>reg_ci_callback()</code> function returns a transient program number of type <code>u_long</code> .				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-level</td><td>Unafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-level	Unafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-level	Unafe				
SEE ALSO	<code>attributes(5)</code>				

NAME	regexpr, compile, step, advance – regular expression compile and match routines
SYNOPSIS	<pre>cc [flag...] [file...] -lgen [library...] #include <regexpr.h> char *compile(char *instring, char *expbuf, const char *endbuf); int step(const char *string, const char *expbuf); int advance(const char *string, const char *expbuf); extern char *loc1, loc2, locs; extern int nbra, regerrno, reglength; extern char *braslist[], *braelist[];</pre>
DESCRIPTION	These routines are used to compile regular expressions and match the compiled expressions against lines. The regular expressions compiled are in the form used by <code>ed(1)</code>. The parameter <i>instring</i> is a null-terminated string representing the regular expression. The parameter <i>expbuf</i> points to the place where the compiled regular expression is to be placed. If <i>expbuf</i> is NULL, <code>compile()</code> uses <code>malloc(3C)</code> to allocate the space for the compiled regular expression. If an error occurs, this space is freed. It is the user's responsibility to free unneeded space after the compiled regular expression is no longer needed. The parameter <i>endbuf</i> is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if <i>expbuf</i> is NULL. If the compiled expression cannot fit in $(endbuf - expbuf)$ bytes, <code>compile()</code> returns NULL and <code>regerrno</code> (see below) is set to 50. The parameter <i>string</i> is a pointer to a string of characters to be checked for a match. This string should be null-terminated. The parameter <i>expbuf</i> is the compiled regular expression obtained by a call of the function <code>compile()</code>. The function <code>step()</code> returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to <code>step()</code>. The variables set in <code>step()</code> are <code>loc1</code> and <code>loc2</code>. <code>loc1</code> is a pointer to the first character that matched the regular expression. The variable <code>loc2</code> points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, <code>loc1</code> points to the first character of <i>string</i> and <code>loc2</code> points to the null at the end of <i>string</i>. The purpose of <code>step()</code> is to step through the <i>string</i> argument until a match is found or until the end of <i>string</i> is reached. If the regular expression begins with <code>^</code>, <code>step()</code> tries to match the regular expression at the beginning of the string only.

regexpr(3GEN)

The `advance()` function is similar to `step()`; but, it only sets the variable `loc2` and always restricts matches to the beginning of the string.

If one is looking for successive matches in the same string of characters, `locs` should be set equal to `loc2`, and `step()` should be called with *string* equal to `loc2`. `locs` is used by commands like `ed` and `sed` so that global substitutions like `s/y*/g` do not loop forever, and is `NULL` by default.

The external variable `nbra` is used to determine the number of subexpressions in the compiled regular expression. `braslist` and `braelist` are arrays of character pointers that point to the start and end of the `nbra` subexpressions in the matched string. For example, after calling `step()` or `advance()` with string `sabcdefg` and regular expression `\(abcdef\)`, `braslist[0]` will point at `a` and `braelist[0]` will point at `g`. These arrays are used by commands like `ed` and `sed` for substitute replacement patterns that contain the `\n` notation for subexpressions.

Note that it is not necessary to use the external variables `regerrno`, `nbra`, `loc1`, `loc2`, `locs`, `braelist`, and `braslist` if one is only checking whether or not a string matches a regular expression.

EXAMPLES

EXAMPLE 1 The following is similar to the regular expression code from `grep`:

```
#include<regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
. . .
if (step(linebuf, expbuf))
    succeed( );
```

RETURN VALUES

If `compile()` succeeds, it returns a non-`NULL` pointer whose value depends on *expbuf*. If *expbuf* is non-`NULL`, `compile()` returns a pointer to the byte after the last byte in the compiled regular expression. The length of the compiled regular expression is stored in `reglength`. Otherwise, `compile()` returns a pointer to the space allocated by `malloc(3C)`.

The functions `step()` and `advance()` return non-zero if the given string matches the regular expression, and zero if the expressions do not match.

ERRORS

If an error is detected when compiling the regular expression, a `NULL` pointer is returned from `compile()` and `regerrno` is set to one of the non-zero error numbers indicated below:

ERROR	MEANING
11	Range endpoint too large.
16	Bad Number.

ERROR	MEANING
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered string search.
42	\(~\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \]&~\}.
45	} expected after \.
46	First number exceeds second in \{~\}.
49	[] imbalance.
50	Regular expression overflow.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(5)

NOTES When compiling multi-threaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-threaded applications.

remainder(3M)

NAME	remainder – remainder function				
SYNOPSIS	<pre>#include <math.h> double remainder(double <i>x</i>, double <i>y</i>);</pre>				
DESCRIPTION	The <code>remainder()</code> function returns the floating point remainder $r = x - ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y. When $n - x/y = 1/2$, the value n is chosen to be even. The behavior of <code>remainder()</code> is independent of the rounding mode.				
RETURN VALUES	The <code>remainder()</code> function returns the floating point remainder $r = x - ny$ when y is non-zero. When y is 0, <code>remainder()</code> returns NaN. and sets <code>errno</code> to <code>EDOM</code>. If the value of x is $\pm\text{Inf}$, <code>remainder()</code> returns NaN and sets <code>errno</code> to <code>EDOM</code>. If x or y is NaN, then the function returns NaN.				
ERRORS	The <code>remainder()</code> function will fail if: <table> <tr> <td><code>EDOM</code></td><td>The y argument is 0 or the x argument is positive or negative infinity.</td></tr> </table>	<code>EDOM</code>	The y argument is 0 or the x argument is positive or negative infinity.		
<code>EDOM</code>	The y argument is 0 or the x argument is positive or negative infinity.				
USAGE	The <code>remainder()</code> function computes the remainder $x \text{ REM } y$ required by ANSI/IEEE 754 (IEC 559).				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>fmod(3M)</code> , <code>attributes(5)</code>				

NAME	rint – round-to-nearest integral value				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double rint(double <i>x</i>);</pre>				
DESCRIPTION	The <code>rint()</code> function returns the integral value (represented as a double) nearest x in the direction of the current IEEE754 rounding mode. If the current rounding mode rounds toward negative infinity, then <code>rint()</code> is identical to <code>floor(3M)</code>. If the current rounding mode rounds toward positive infinity, then <code>rint()</code> is identical to <code>ceil(3M)</code>.				
RETURN VALUES	Upon successful completion, the <code>rint()</code> function returns the integer (represented as a double precision number) nearest x in the direction of the current IEEE754 rounding mode. When x is $\pm\text{Inf}$, <code>rint()</code> returns x. If the value of x is NaN, NaN is returned.				
ERRORS	No errors will occur.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>ceil(3M)</code> , <code>floor(3M)</code> , <code>isnan(3M)</code> , <code>attributes(5)</code>				

rsm_create_localmemory_handle(3RSM)

NAME	rsm_create_localmemory_handle, rsm_free_localmemory_handle – create or free local memory handle										
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_create_localmemory_handle(rsmapi_controller_handle_t handle, rsm_localmemory_handle_t *l_handle, caddr_t local_vaddr, size_t length); int rsm_free_localmemory_handle(rsmapi_controller_handle_t handle, rsm_localmemory_handle_t l_handle);</pre>										
DESCRIPTION	The <code>rsm_create_localmemory_handle()</code> and <code>rsm_free_localmemory_handle()</code> functions are supporting functions for <code>rsm_memseg_import_putv(3RSM)</code> and <code>rsm_memseg_import_getv(3RSM)</code>. The <code>rsm_create_localmemory_handle()</code> function creates a local memory handle to be used in the I/O vector component of a scatter-gather list of subsequent <code>rsm_memseg_import_putv()</code> and <code>rsm_memseg_import_getv()</code> calls. The <i>handle</i> argument specifies the controller handle obtained from <code>rsm_get_controller(3RSM)</code>. The <i>l_handle</i> argument is a pointer to the location for the function to return the local memory handle. The <i>local_vaddr</i> argument specifies the local virtual address; it should be aligned at a page boundary. The <i>length</i> argument specifies the length of memory spanned by the handle. The <code>rsm_free_localmemory_handle()</code> function unlocks the memory range for the local handle specified by <i>l_handle</i> and releases the associated system resources. The <i>handle</i> argument specifies the controller handle. All handles created by a process are freed when the process exits, but the process should call <code>rsm_free_localmemory_handle()</code> as soon as possible to free the system resources.										
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.										
ERRORS	The <code>rsm_create_localmemory_handle()</code> and <code>rsm_free_localmemory_handle()</code> functions can return the following errors: <table> <tr> <td>RSMERR_BAD_CTLR_HNDL</td><td>Invalid controller handle.</td></tr> <tr> <td>RSMERR_BAD_LOCALMEM_HNDL</td><td>Invalid local memory handle.</td></tr> </table> The <code>rsm_create_localmemory_handle()</code> function can return the following errors: <table> <tr> <td>RSMERR_BAD_LENGTH</td><td>Invalid length.</td></tr> <tr> <td>RSMERR_BAD_ADDRESS</td><td>Invalid address.</td></tr> <tr> <td>RSMERR_INSUFFICIENT_MEM</td><td>Insufficient memory.</td></tr> </table>	RSMERR_BAD_CTLR_HNDL	Invalid controller handle.	RSMERR_BAD_LOCALMEM_HNDL	Invalid local memory handle.	RSMERR_BAD_LENGTH	Invalid length.	RSMERR_BAD_ADDRESS	Invalid address.	RSMERR_INSUFFICIENT_MEM	Insufficient memory.
RSMERR_BAD_CTLR_HNDL	Invalid controller handle.										
RSMERR_BAD_LOCALMEM_HNDL	Invalid local memory handle.										
RSMERR_BAD_LENGTH	Invalid length.										
RSMERR_BAD_ADDRESS	Invalid address.										
RSMERR_INSUFFICIENT_MEM	Insufficient memory.										

`rsm_create_localmemory_handle(3RSM)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_import_putv(3RSM)`, `attributes(5)`

rsm_get_controller(3RSM)

NAME	rsm_get_controller, rsm_get_controller_attr, rsm_release_controller – get or release a controller handle										
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_get_controller(char *name, rsmapi_controller_handle_t *controller) ; int rsm_get_controller_attr(rsmapi_controller_handle_t chdl, rsmapi_controller_attr_t *attr) ; int rsm_release_controller(rsmapi_controller_handle_t chdl) ;</pre>										
DESCRIPTION	The controller functions provide mechanisms for obtaining access to a controller, determining the characteristics of the controller, and releasing the controller. The <code>rsm_get_controller()</code> function acquires a controller handle through the <i>controller</i> argument. The <i>name</i> argument is the specific controller instance (for example, "sci0" or "loopback"). This controller handle is used for subsequent RSMAPI calls. The <code>rsm_get_controller_attr()</code> function obtains a controller's attributes through the <i>attr</i> argument. The <i>chdl</i> argument is the controller handle obtained by the <code>rsm_get_controller()</code> call. The attribute structure is defined in the <code><rsmapi></code> header. The <code>rsm_release_controller()</code> function releases the resources associated with the controller identified by the controller handle <i>chdl</i>, obtained by calling <code>rsm_get_controller()</code>. Each <code>rsm_release_controller()</code> call must have a corresponding <code>rsm_get_controller()</code> call. It is illegal to access a controller or segments exported or imported using a released controller.										
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.										
ERRORS	The <code>rsm_get_controller()</code>, <code>rsm_get_controller_attr()</code>, and <code>rsm_release_controller()</code> functions can return the following errors: <table> <tr> <td>RSMERR_BAD_CTLR_HNDL</td><td>Invalid controller handle.</td></tr> </table> The <code>rsm_get_controller()</code> and <code>rsm_get_controller_attr()</code> functions can return the following errors: <table> <tr> <td>RSMERR_BAD_ADDR</td><td>Bad address.</td></tr> </table> The <code>rsm_get_controller()</code> function can return the following errors: <table> <tr> <td>RSMERR_CTLR_NOT_PRESENT</td><td>Controller not present.</td></tr> <tr> <td>RSMERR_INSUFFICIENT_MEM</td><td>Insufficient memory.</td></tr> <tr> <td>RSMERR_BAD_LIBRARY_VERSION</td><td>Invalid library version.</td></tr> </table>	RSMERR_BAD_CTLR_HNDL	Invalid controller handle.	RSMERR_BAD_ADDR	Bad address.	RSMERR_CTLR_NOT_PRESENT	Controller not present.	RSMERR_INSUFFICIENT_MEM	Insufficient memory.	RSMERR_BAD_LIBRARY_VERSION	Invalid library version.
RSMERR_BAD_CTLR_HNDL	Invalid controller handle.										
RSMERR_BAD_ADDR	Bad address.										
RSMERR_CTLR_NOT_PRESENT	Controller not present.										
RSMERR_INSUFFICIENT_MEM	Insufficient memory.										
RSMERR_BAD_LIBRARY_VERSION	Invalid library version.										

`rsm_get_controller(3RSM)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_export_create(3RSM)`, `rsm_memseg_import_connect(3RSM)`,
`attributes(5)`

rsm_get_interconnect_topology(3RSM)

NAME	rsm_get_interconnect_topology, rsm_free_interconnect_topology – get or free interconnect topology						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_get_interconnect_topology(rsm_topology_t **topology_data); void rsm_free_interconnect_topology(rsm_topology_t *topology_data);</pre>						
DESCRIPTION	The <code>rsm_get_interconnect_topology(3RSM)</code> and <code>rsm_free_interconnect_topology(3RSM)</code> functions provide for access to the interconnect controller and connection data. The key interconnect data required for export and import operations includes the respective cluster nodeids and the controller names. To facilitate applications in the establishment of proper and efficient export and import policies, a delineation of the interconnect topology is provided by this interface. The data provided includes local nodeid, local controller name, its hardware address, and remote connection specification for each local controller. An application component exporting memory can thus find the set of existing local controllers and correctly assign controllers for the creation and publishing of segments. Exported segments may also be efficiently distributed over the set of controllers consistent with the hardware interconnect and application software. An application component which is to import memory must be informed of the segment id(s) and controller(s) used in the exporting of memory, this needs to be done using some out-of-band mechanism. The topology data structures are defined in the <code><rsmapi.h></code> header. The <code>rsm_get_interconnect_topology()</code> returns a pointer to the topology data in a location specified by the <code>topology_data</code> argument. The <code>rsm_free_interconnect_topology()</code> frees the resources allocated by <code>rsm_get_interconnect_topology()</code>.						
RETURN VALUES	Upon successful completion, <code>rsm_get_interconnect_topology()</code> returns 0. Otherwise, an error value is returned to indicate the error.						
ERRORS	The <code>rsm_get_interconnect_topology()</code> function can return the following errors: <table><tr><td>RSMERR_BAD_TOPOLOGY_PTR</td><td>Invalid topology pointer.</td></tr><tr><td>RSMERR_INSUFFICIENT_MEM</td><td>Insufficient memory.</td></tr><tr><td>RSMERR_BAD_ADDR</td><td>Bad address.</td></tr></table>	RSMERR_BAD_TOPOLOGY_PTR	Invalid topology pointer.	RSMERR_INSUFFICIENT_MEM	Insufficient memory.	RSMERR_BAD_ADDR	Bad address.
RSMERR_BAD_TOPOLOGY_PTR	Invalid topology pointer.						
RSMERR_INSUFFICIENT_MEM	Insufficient memory.						
RSMERR_BAD_ADDR	Bad address.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Interface Stability</td><td>Evolving</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						

rsm_get_interconnect_topology(3RSM)

MT-Level	MT-Safe

SEE ALSO attributes(5)

rsm_get_segmentid_range(3RSM)

NAME	rsm_get_segmentid_range – get segment ID range						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_get_segmentid_range(const char *appid, rsm_segment_id_t *baseid, uint_t *length);</pre>						
DESCRIPTION	RSM segment IDs can be either specified by the application or generated by the system using the <code>rsm_memseg_export_publish(3RSM)</code> function. Applications that specify segment IDs require a reserved range of segment IDs that they can use. This can be achieved by using <code>rsm_get_segmentid_range()</code> and by reserving a range of segment IDs in the segment ID configuration file, <code>/etc/rsm/rsm.segmentid</code>. The <code>rsm_get_segmentid_range()</code> function can be used by applications to obtain the segment ID range reserved for them. The <i>appid</i> argument is a null-terminated string that identifies the application. The <i>baseid</i> argument points to the location where the starting segment ID of the reserved range is returned. The <i>length</i> argument points to the location where the number of reserved segment IDs is returned. The application can use any value starting at <i>baseid</i> and less than <i>baseid+length</i>. The application should use an offset within the range of reserved segment IDs to obtain a segment ID such that if the <i>baseid</i> or <i>length</i> is modified, it will still be within its reserved range. It is the responsibility of the system administrator to make sure that the segment ID ranges are properly administered (such that they are non-overlapping, the file on various nodes of the cluster have identical entries, and so forth.) Entries in the <code>/etc/rsm/rsm.segmentid</code> file are of the form: <pre>#keyword appid baseid length reserve SUNWfoo 0x600000 1000</pre> The fields in the file are separated by tabs or blanks. The first string is a keyword "reserve", followed by the application identifier (a string without spaces), the <i>baseid</i> (the starting segment ID of the reserved range in hexadecimal), and the <i>length</i> (the number of segmentids reserved). Comment lines contain a "#" in the first column. The file should not contain blank or empty lines. Segment IDs reserved for the system are defined in the <code></usr/include/rsm/rsm_common.h></code> header and cannot be used by the applications.						
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.						
ERRORS	The <code>rsm_get_segmentid_range()</code> function can return the following errors: <table> <tr> <td>RSMERR_BAD_ADDR</td><td>The address passed is invalid.</td></tr> <tr> <td>RSMERR_BAD_APPID</td><td>The <i>appid</i> is not defined in configuration file.</td></tr> <tr> <td>RSMERR_BAD_CONF</td><td>The configuration file is not present or not readable, or the configuration file format is incorrect.</td></tr> </table>	RSMERR_BAD_ADDR	The address passed is invalid.	RSMERR_BAD_APPID	The <i>appid</i> is not defined in configuration file.	RSMERR_BAD_CONF	The configuration file is not present or not readable, or the configuration file format is incorrect.
RSMERR_BAD_ADDR	The address passed is invalid.						
RSMERR_BAD_APPID	The <i>appid</i> is not defined in configuration file.						
RSMERR_BAD_CONF	The configuration file is not present or not readable, or the configuration file format is incorrect.						

rsm_get_segmentid_range(3RSM)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable
MT-Level	MT-Safe

SEE ALSO rsm_memseg_export_publish(3RSM), attributes(5)

rsm_intr_signal_post(3RSM)

NAME	rsm_intr_signal_post, rsm_intr_signal_wait – signal or wait for an event								
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_intr_signal_post(void *memseg, uint_t flags); int rsm_intr_signal_wait(void *memseg, int timeout);</pre>								
DESCRIPTION	The <code>rsm_intr_signal_post()</code> and <code>rsm_intr_signal_wait()</code> functions are event functions that allow synchronization between importer processes and exporter processes. A process may block to wait for an event occurrence by calling <code>rsm_intr_signal_wait()</code>. A process can signal a waiting process when an event occurs by calling <code>rsm_intr_signal_post()</code>. The <code>rsm_intr_signal_post()</code> function signals an event occurrence. Either an import segment handle (<code>rsm_memseg_import_handle_t</code>) or an export segment handle (<code>rsm_memseg_export_handle_t</code>) may be type cast to a void pointer for the <i>memseg</i> argument. If <i>memseg</i> refers to an import handle, the exporting process is signalled. If <i>memseg</i> refers to an export handle, all importers of that segment are signalled. The <i>flags</i> argument may be set to <code>RSM_SIGPOST_NO_ACCUMULATE</code>; this will cause this event to be discarded if an event is already pending for the target segment. The <code>rsm_intr_signal_wait()</code> function allows a process to block and wait for an event occurrence. Either an import segment handle (<code>rsm_memseg_import_handle_t</code>) or an export segment handle (<code>rsm_memseg_export_handle_t</code>) may be type cast to a void pointer for the <i>memseg</i> argument. The process blocks for up to <i>timeout</i> milliseconds for an event to occur; if the timeout value is -1, the process blocks until an event occurs or until interrupted.								
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.								
ERRORS	The <code>rsm_intr_signal_post()</code> and <code>rsm_intr_signal_wait()</code> functions can return the following error: <table><tr><td>RSMERR_BAD_SEG_HNDL</td><td>Invalid segment handle.</td></tr></table> The <code>rsm_intr_signal_post()</code> function can return the following error: <table><tr><td>RSMERR_REMOTE_NODE_UNREACHABL</td><td>Remote node not reachable.</td></tr></table> The <code>rsm_intr_signal_wait()</code> function can return the following errors: <table><tr><td>RSMERR_TIMEOUT</td><td>Timer expired.</td></tr><tr><td>RSMERR_INTERRUPTED</td><td>Wait interrupted.</td></tr></table>	RSMERR_BAD_SEG_HNDL	Invalid segment handle.	RSMERR_REMOTE_NODE_UNREACHABL	Remote node not reachable.	RSMERR_TIMEOUT	Timer expired.	RSMERR_INTERRUPTED	Wait interrupted.
RSMERR_BAD_SEG_HNDL	Invalid segment handle.								
RSMERR_REMOTE_NODE_UNREACHABL	Remote node not reachable.								
RSMERR_TIMEOUT	Timer expired.								
RSMERR_INTERRUPTED	Wait interrupted.								

`rsm_intr_signal_post(3RSM)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_get_pollfd(3RSM)`, `attributes(5)`

rsm_memseg_export_create(3RSM)

NAME	rsm_memseg_export_create, rsm_memseg_export_destroy, rsm_memseg_export_rebind – resource allocation and management functions for export memory segments
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_export_create(rsmapi_controller_handle_t controller, rsm_memseg_export_handle_t *memseg, void *vaddr, size_t length, uint_t flags); int rsm_memseg_export_destroy(rsm_memseg_export_handle_t memseg); int rsm_memseg_export_rebind(rsm_memseg_export_handle_t memseg, void *vaddr, offset_t off, size_t length);</pre>
DESCRIPTION	The <code>rsm_memseg_export_create()</code>, <code>rsm_memseg_export_destroy()</code>, and <code>rsm_memseg_export_rebind()</code> functions provide for allocation and management of resources supporting export memory segments. Exporting a memory segment involves the application allocating memory in its virtual address space through the System V shared memory interface or normal operating system memory allocation functions. This is followed by the calls to create the export segment and bind physical pages to back to allocated virtual address space. The <code>rsm_memseg_export_create()</code> creates a new memory segment. Physical memory pages are allocated and are associated with the segment. The segment lifetime is the same as the lifetime of the creating process or until a destroy operation is performed. The <i>controller</i> argument is the controller handle obtained from a prior call to <code>rsm_get_controller(3RSM)</code>. The export memory segment handle is obtained through the <i>memseg</i> argument for use in subsequent operations. The <i>vaddr</i> argument specifies the process virtual address for the segment. It must be aligned according to the controller page size attribute. The <i>length</i> argument specifies the size of the segment in bytes and must be in multiples of the controller page size. The <i>flags</i> argument is a bitmask of flags. The <code>RSM_ALLOW_REBIND</code> flag indicates that unbind and rebind is allowed on the segment during its lifetime. The <code>RSM_LOCK_OPS</code> flag indicates that this segment can be used for lock operations. The <code>rsm_memseg_export_destroy()</code> function deallocates the physical memory pages associated with the segment and disconnects all importers of the segment. The <i>memseg</i> argument is the export memory segment handle obtained by a call to <code>rsm_memseg_export_create()</code>. The <code>rsm_memseg_export_rebind()</code> function releases the current backing pages associated with the segment and allocates new physical memory pages. This operation is transparent to the importers of the segment. It is the responsibility of the application to prevent data access to the export segment until the rebind operation has completed. Segment data access during rebind does not cause a system failure but data content results are undefined. The <i>memseg</i> argument is the export segment handle pointer obtained from <code>rsm_memseg_export_create()</code>. The <i>vaddr</i> argument must be aligned with respect to the page size attribute of the controller. The <i>length</i> argument modulo controller page size must be 0. The <i>off</i> argument is currently unused.

RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.																										
ERRORS	The <code>rsm_memseg_export_create()</code>, <code>rsm_memseg_export_destroy()</code>, and <code>rsm_memseg_export_rebind()</code> functions can return the following errors: <table> <tr> <td><code>RSMERR_BAD_SEG_HNDL</code></td><td>Invalid segment handle.</td></tr> </table> The <code>rsm_memseg_export_create()</code> and <code>rsm_memseg_export_rebind()</code> functions can return the following errors: <table> <tr> <td><code>RSMERR_BAD_CTLR_HNDL</code></td><td>Invalid controller handle.</td></tr> <tr> <td><code>RSMERR_CTLR_NOT_PRESENT</code></td><td>Controller not present.</td></tr> <tr> <td><code>RSMERR_BAD_LENGTH</code></td><td>Length zero or length exceeds controller limits.</td></tr> <tr> <td><code>RSMERR_BAD_ADDR</code></td><td>Invalid address.</td></tr> <tr> <td><code>RSMERR_INSUFFICIENT_MEM</code></td><td>Insufficient memory.</td></tr> <tr> <td><code>RSMERR_INSUFFICIENT_RESOURCES</code></td><td>Insufficient resources.</td></tr> <tr> <td><code>RSMERR_PERM_DENIED</code></td><td>Permission denied.</td></tr> <tr> <td><code>RSMERR_NOT_CREATOR</code></td><td>Not creator of segment.</td></tr> <tr> <td><code>RSMERR_REBIND_NOT_ALLOWED</code></td><td>Rebind not allowed.</td></tr> </table> The <code>rsm_memseg_export_create()</code> function can return the following errors: <table> <tr> <td><code>RSMERR_BAD_MEM_ALIGNMENT</code></td><td>The address is not aligned on a page boundary.</td></tr> </table> The <code>rsm_memseg_export_rebind()</code> function can return the following errors: <table> <tr> <td><code>RSMERR_INTERRUPTED</code></td><td>The operation was interrupted by a signal.</td></tr> </table> The <code>rsm_memseg_export_destroy()</code> function can return the following errors: <table> <tr> <td><code>RSMERR_POLLFD_IN_USE</code></td><td>Poll file descriptor in use.</td></tr> </table>	<code>RSMERR_BAD_SEG_HNDL</code>	Invalid segment handle.	<code>RSMERR_BAD_CTLR_HNDL</code>	Invalid controller handle.	<code>RSMERR_CTLR_NOT_PRESENT</code>	Controller not present.	<code>RSMERR_BAD_LENGTH</code>	Length zero or length exceeds controller limits.	<code>RSMERR_BAD_ADDR</code>	Invalid address.	<code>RSMERR_INSUFFICIENT_MEM</code>	Insufficient memory.	<code>RSMERR_INSUFFICIENT_RESOURCES</code>	Insufficient resources.	<code>RSMERR_PERM_DENIED</code>	Permission denied.	<code>RSMERR_NOT_CREATOR</code>	Not creator of segment.	<code>RSMERR_REBIND_NOT_ALLOWED</code>	Rebind not allowed.	<code>RSMERR_BAD_MEM_ALIGNMENT</code>	The address is not aligned on a page boundary.	<code>RSMERR_INTERRUPTED</code>	The operation was interrupted by a signal.	<code>RSMERR_POLLFD_IN_USE</code>	Poll file descriptor in use.
<code>RSMERR_BAD_SEG_HNDL</code>	Invalid segment handle.																										
<code>RSMERR_BAD_CTLR_HNDL</code>	Invalid controller handle.																										
<code>RSMERR_CTLR_NOT_PRESENT</code>	Controller not present.																										
<code>RSMERR_BAD_LENGTH</code>	Length zero or length exceeds controller limits.																										
<code>RSMERR_BAD_ADDR</code>	Invalid address.																										
<code>RSMERR_INSUFFICIENT_MEM</code>	Insufficient memory.																										
<code>RSMERR_INSUFFICIENT_RESOURCES</code>	Insufficient resources.																										
<code>RSMERR_PERM_DENIED</code>	Permission denied.																										
<code>RSMERR_NOT_CREATOR</code>	Not creator of segment.																										
<code>RSMERR_REBIND_NOT_ALLOWED</code>	Rebind not allowed.																										
<code>RSMERR_BAD_MEM_ALIGNMENT</code>	The address is not aligned on a page boundary.																										
<code>RSMERR_INTERRUPTED</code>	The operation was interrupted by a signal.																										
<code>RSMERR_POLLFD_IN_USE</code>	Poll file descriptor in use.																										
USAGE	Exporting a memory segment involves the application allocating memory in its virtual address space through the System V Shared Memory interface or other normal operating system memory allocation methods such as <code>valloc()</code> (see <code>malloc(3C)</code>) or <code>mmap(2)</code> . Memory for a file mapped with <code>mmap()</code> must be mapped <code>MAP_PRIVATE</code> .																										
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:																										

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Unstable

rsm_memseg_export_create(3RSM)

MT-Level	MT-Safe

SEE ALSO rsm_get_controller(3RSM), rsm_memseg_export_publish(3RSM),
attributes(5)

	rsm_memseg_export_publish(3RSM)						
NAME	rsm_memseg_export_publish, rsm_memseg_export_unpublish, rsm_memseg_export_republish – allow or disallow a memory segment to be imported by other nodes						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_export_publish(rsm_memseg_export_handle_t memseg, rsm_memseg_id_t *segment_id, rsmapi_access_entry_t access_list[], uint_t access_list_length); int rsm_memseg_export_unpublish(rsm_memseg_export_handle_t memseg); int rsm_memseg_export_republish(rsm_memseg_export_handle_t memseg, rsmapi_access_entry_t access_list[], uint_t access_list_length);</pre>						
DESCRIPTION	The <code>rsm_memseg_export_publish()</code>, <code>rsm_memseg_export_unpublish()</code>, and <code>rsm_memseg_export_republish()</code> functions allow or disallow a memory segment to be imported by other nodes. The <code>rsm_memseg_export_publish(3RSM)</code> function allows the export segment specified by the <i>memseg</i> argument to be imported by other nodes. It also assigns a unique segment identifier to the segment and defines the access control list for the segment. The <i>segment_id</i> argument is a pointer to an identifier which is unique on the publishing node. It is the responsibility of the application to manage the assignment of unique segment identifiers. The identifier can be optionally initialized to 0, in which case the system will return a unique segment identifier value. The <i>access_list</i> argument is composed of pairs of nodeid and access permissions. For each nodeid specified in the list, the associated read/write permissions are provided by three octal digits for owner, group, and other, as for Solaris file permissions. In the access control each octal digit may have the following values: <table> <tr> <td>2</td><td>write access</td></tr> <tr> <td>4</td><td>read only access</td></tr> <tr> <td>6</td><td>read and write access</td></tr> </table> An access permissions value of 0624 specifies: (1) an importer with the same uid as the exporter has read and write access; (2) an importer with the same gid as the exporter has write access only; and (3) all other importers have read access only. When an access control list is provided, nodes not included in the list will be prevented from importing the segment. However, if the access list is NULL (this will require the length <i>access_list_length</i> to be specified as 0 as well), then no nodes will be excluded from importing and the access permissions on all nodes will equal the owner-group-other file creation permissions of the exporting process. Corresponding to the <i>access_list</i> argument, the <i>access_list_length</i> argument specifies the number of entries in the <i>access_list</i> array.	2	write access	4	read only access	6	read and write access
2	write access						
4	read only access						
6	read and write access						

rsm_memseg_export_publish(3RSM)

The `rsm_memseg_export_unpublish()` function disallows the export segment specified by *memseg* from being imported. All the existing import connections are forcibly disconnected.

The `rsm_memseg_export_republish()` function changes the access control list for the exported and published segment. Although the current import connections remain unaffected by this call, new connections are constrained by the new access list.

RETURN VALUES

Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

ERRORS

The `rsm_memseg_export_publish()`, `rsm_memseg_export_unpublish()`, and `rsm_memseg_export_republish()` functions can return the following errors:

RSMERR_BAD_SEG_HNDL Invalid segment handle.

RSMERR_NOT_CREATOR Not creator of segment.

The `rsm_memseg_export_publish()` and `rsm_memseg_export_republish()` functions can return the following errors, with the exception that only `rsm_memseg_export_publish()` can return the errors related to the segment identifier:

RSMERR_SEGID_IN_USE Segment identifier in use.

RSMERR_RESERVED_SEGID Segment identifier reserved.

RSMERR_BAD_SEGID Invalid segment identifier.

RSMERR_BAD_ACL Invalid access control list.

RSMERR_SEG_ALREADY_PUBLISHED Segment already published.

RSMERR_INSUFFICIENT_MEM Insufficient memory.

RSMERR_INSUFFICIENT_RESOURCES Insufficient resources.

RSMERR_LOCKS_NOT_SUPPORTED Locks not supported.

RSMERR_BAD_ADDR Bad address.

The `rsm_memseg_export_republish()` and `rsm_memseg_export_unpublish()` functions can return the following errors:

RSMERR_SEG_NOT_PUBLISHED Segment not published.

RSMERR_INTERRUPTED The operation was interrupted by a signal.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

rsm_memseg_export_publish(3RSM)

MT-Level	MT-Safe

SEE ALSO rsm_memseg_export_create(3RSM), attributes(5)

rsm_memseg_get_pollfd(3RSM)

NAME	rsm_memseg_get_pollfd, rsm_memseg_release_pollfd – get or release a poll descriptor						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_get_pollfd(void *memseg, struct pollfd *fd); int rsm_memseg_release_pollfd(void *memseg);</pre>						
DESCRIPTION	The <code>rsm_memseg_get_pollfd()</code> and <code>rsm_memseg_release_pollfd()</code> functions provide an alternative to <code>rsm_intr_signal_wait(3RSM)</code>; the waiting process may multiplex event waiting using the <code>poll(2)</code> function after first obtaining a poll descriptor using <code>rsm_memseg_get_pollfd()</code>. The descriptor may subsequently be released using <code>rsm_memseg_release_pollfd()</code>. As a result of a call <code>rsm_memseg_get_pollfd()</code>, the specified <code>pollfd</code> structure is initialized with a descriptor for the specified segment (<i>memseg</i>) and the event generated by <code>rsm_intr_signal_post(3RSM)</code>. Either an export segment handle or an import segment handle may be type cast to a void pointer. The <i>pollfd</i> argument may subsequently be used with the <code>poll(2)</code> function to wait for the event. If <i>memseg</i> references an export segment, the segment must be currently published. If <i>memseg</i> references an import segment, the segment must be connected. The <code>rsm_memseg_release_pollfd()</code> function decrements the reference count of the <code>pollfd</code> structure associated with the specified segment. A segment unpublish, destroy or unmap operation will fail if the reference count is non-zero.						
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.						
ERRORS	The <code>rsm_memseg_get_pollfd()</code> and <code>rsm_memseg_release_pollfd()</code> function can return the following errors: RSMERR_BAD_SEG_HNDL Invalid segment handle.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>poll(2)</code> , <code>rsm_intr_signal_post(3RSM)</code> , <code>attributes(5)</code>						

rsm_memseg_import_connect(3RSM)

NAME	rsm_memseg_import_connect, rsm_memseg_import_disconnect – create or break logical connection between import and export segments								
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_connect(rsmapi_controller_handle_t controller, rsm_node_id_t nodeid, rsm_memseg_id_t segment_id, rsm_permission_t perm, rsm_memseg_import_handle_t *memseg); int rsm_memseg_import_disconnect(rsm_memseg_import_handle_t memseg);</pre>								
DESCRIPTION	The <code>rsm_memseg_import_connect()</code> function provides a means of creating an import segment called <i>memseg</i> and establishing a logical connection with an export segment identified by the <i>segment_id</i> on the node specified by <i>node_id</i>. The controller specified by <i>controller</i> must have a physical connection with the controller (see <code>rsm_get_interconnect_topology(3RSM)</code>) used while exporting the segment identified by <i>segment_id</i> on node specified by <i>node_id</i>. The <i>perm</i> argument specifies the mode of access that the importer is requesting for this connection. In the connection process, the mode of access and the importers userid and groupid are compared with the access permissions specified by the exporter. If the request mode is not valid, the connection request is denied. The <i>perm</i> argument is limited to the following octal values: <table style="margin-left: 20px;"> <tr><td>0400</td><td>read mode</td></tr> <tr><td>0200</td><td>write mode</td></tr> <tr><td>0600</td><td>read/write mode</td></tr> </table> The <code>rsm_memseg_import_disconnect()</code> function breaks the logical connection between the import segment and the exported segment and deallocates the resources associated with the import segment handle <i>memseg</i>.	0400	read mode	0200	write mode	0600	read/write mode		
0400	read mode								
0200	write mode								
0600	read/write mode								
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.								
ERRORS	The <code>rsm_memseg_import_connect()</code> and <code>rsm_memseg_import_disconnect()</code> functions can return the following errors: <table style="margin-left: 20px;"> <tr> <td>RSMERR_BAD_SEG_HNDL</td> <td>Invalid segment handle.</td> </tr> </table> The <code>rsm_memseg_import_connect()</code> function can return the following errors: <table style="margin-left: 20px;"> <tr> <td>RSMERR_BAD_CTLR_HNDL</td> <td>Invalid controller handle.</td> </tr> <tr> <td>RSMERR_CTLR_NOT_PRESENT</td> <td>Controller not present.</td> </tr> <tr> <td>RSMERR_PERM_DENIED</td> <td>Permission denied.</td> </tr> </table>	RSMERR_BAD_SEG_HNDL	Invalid segment handle.	RSMERR_BAD_CTLR_HNDL	Invalid controller handle.	RSMERR_CTLR_NOT_PRESENT	Controller not present.	RSMERR_PERM_DENIED	Permission denied.
RSMERR_BAD_SEG_HNDL	Invalid segment handle.								
RSMERR_BAD_CTLR_HNDL	Invalid controller handle.								
RSMERR_CTLR_NOT_PRESENT	Controller not present.								
RSMERR_PERM_DENIED	Permission denied.								

rsm_memseg_import_connect(3RSM)

RSMERR_INSUFFICIENT_MEM
Insufficient memory.

RSMERR_INSUFFICIENT_RESOURCES
Insufficient resources.

RSMERR_SEG_NOT_PUBLISHED_TO_NODE
Segment not published to node.

RSMERR_SEG_NOT_PUBLISHED
Segment not published at all.

RSMERR_BAD_ADDR
Bad address.

RSMERR_REMOTE_NODE_UNREACHABLE
Remote not not reachable.

RSMERR_INTERRUPTED
Connection interrupted.

The rsm_memseg_import_disconnect() function can return the following errors:

RSMERR_SEG_STILL_MAPPED Segment still mapped, need to unmap
before disconnect.

RSMERR_POLLFD_IN_USE Poll file descriptor in use.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO rsm_memseg_import_map(3RSM), attributes(5)

NAME	rsm_memseg_import_get, rsm_memseg_import_get8, rsm_memseg_import_get16, rsm_memseg_import_get32, rsm_memseg_import_get64 – read from a segment
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_get(rsm_memseg_import_handle_t im_memseg, off_t offset, void *dest_addr, size_t length); int rsm_memseg_import_get8(rsm_memseg_import_handle_t im_memseg, off_t offset, uint8_t *datap, ulong_t rep_cnt); int rsm_memseg_import_get16(rsm_memseg_import_handle_t im_memseg, off_t offset, uint16_t *datap, ulong_t rep_cnt); int rsm_memseg_import_get32(rsm_memseg_import_handle_t im_memseg, off_t offset, uint32_t *datap, ulong_t rep_cnt); int rsm_memseg_import_get64(rsm_memseg_import_handle_t im_memseg, off_t offset, uint64_t *datap, ulong_t rep_cnt);</pre>
DESCRIPTION	When using interconnects that allow memory mapping (see <code>rsm_memseg_import_map(3RSM)</code>), standard CPU memory operations may be used for accessing memory of a segment. If a mapping is not provided, then explicitly calling these functions facilitates reading from a segment. Depending on the attributes of the extension library of the specific interconnect, these functions may involve performing an implicit mapping before performing the data transfer. Applications can be made interconnect-independent with respect to segment reads by using these functions. The data access error detection is performed through the use of barriers (see <code>rsm_memseg_import_open_barrier(3RSM)</code>). The default barrier operation mode is <code>RSM_BARRIER_MODE_IMPLICIT</code>, meaning that around every get operation open and close barrier are performed automatically. Alternatively, explicit error handling may be set up for these functions (see <code>rsm_memseg_import_set_mode(3RSM)</code>). In either case the barrier should be initialized prior to using these functions using <code>rsm_memseg_import_init_barrier(3RSM)</code>. The <code>rsm_memseg_import_get()</code> function copies <i>length</i> bytes from the imported segment <i>im_memseg</i> beginning at location <i>offset</i> from the start of the segment to a local memory buffer pointed to by <i>dest_addr</i>. The <code>rsm_memseg_import_get8()</code> function copies <i>rep_cnt</i> number of 8-bit quantities from successive locations starting from <i>offset</i> in the imported segment to successive local memory locations pointed to by <i>datap</i>. The <code>rsm_memseg_import_get16()</code> functions copies <i>rep_cnt</i> number of 16-bit quantities from successive locations starting from <i>offset</i> in the imported segment to successive local memory locations pointed to by <i>datap</i>. The offset must be aligned at half-word address boundary.

rsm_memseg_import_get(3RSM)

The `rsm_memseg_import_get32()` function copies *rep_cnt* number of 32-bit quantities from successive locations starting from *offset* in the imported segment to successive local memory locations pointed to by *datap*. The offset must be aligned at word address boundary.

The `rsm_memseg_import_get64()` function copies *rep_cnt* number of -bit quantities from successive locations starting from *offset* in the imported segment to successive local memory locations pointed to by *datap*. The offset must be aligned at double-word address boundary.

The data transfer functions that transfer small quantities of data (that is, 8-, 16-, 32-, and 64-bit quantities) perform byte swapping prior to the data transfer, in the event that the source and destination have incompatible endian characteristics.

RETURN VALUES Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

ERRORS These functions can return the following errors:

RSMERR_BAD_SEG_HNDL	Invalid segment handle.
RSMERR_BAD_ADDR	Bad address.
RSMERR_BAD_MEM_ALIGNMENT	Invalid memory alignment for pointer.
RSMERR_BAD_OFFSET	Invalid offset.
RSMERR_BAD_LENGTH	Invalid length.
RSMERR_PERM_DENIED	Permission denied.
RSMERR_INSUFFICIENT_RESOURCES	Insufficient resources.
RSMERR_BARRIER_UNINITIALIZED	Barrier not initialized.
RSMERR_BARRIER_FAILURE	I/O completion error.
RSMERR_CONN_ABORTED	Connection aborted.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_import_init_barrier(3RSM)`,
`rsm_memseg_import_open_barrier(3RSM)`,
`rsm_memseg_import_set_mode(3RSM)`, `attributes(5)`

rsm_memseg_import_init_barrier(3RSM)

NAME	rsm_memseg_import_init_barrier, rsm_memseg_import_destroy_barrier – create or destroy barrier for imported segment						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_init_barrier(rsm_memseg_import_handle_t memseg, rsm_barrier_type_t type, rsmapi_barrier_t *barrier); int rsm_memseg_import_destroy_barrier(rsmapi_barrier_t *barrier);</pre>						
DESCRIPTION	The <code>rsm_memseg_import_init_barrier()</code> function creates a barrier for the imported segment specified by <i>memseg</i>. The barrier type is specified by the <i>type</i> argument. Currently, only <code>RSM_BAR_DEFAULT</code> is supported as a barrier type. A handle to the barrier is obtained through the <i>barrier</i> argument and is used in subsequent barrier calls. The <code>rsm_memseg_import_destroy_barrier()</code> function deallocates all the resources associated with the barrier.						
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.						
ERRORS	The <code>rsm_memseg_import_init_barrier()</code> and <code>rsm_memseg_import_destroy_barrier()</code> functions can return the following errors: <code>RSMERR_BAD_SEG_HNDL</code> Invalid segment handle. <code>RSMERR_BAD_BARRIER_PTR</code> Invalid barrier pointer. The <code>rsm_memseg_import_init_barrier()</code> function can return the following errors: <code>RSMERR_INSUFFICIENT_MEM</code> Insufficient memory.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>rsm_memseg_import_open_barrier(3RSM)</code>, <code>rsm_memseg_import_set_mode(3RSM)</code>, <code>attributes(5)</code>						

rsm_memseg_import_map(3RSM)

NAME	rsm_memseg_import_map, rsm_memseg_import_unmap – map or unmap imported segment
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_map(rsm_memseg_import_handle_t im_memseg, void **address, rsm_attribute_t attr, rsm_permission_t perm, off_t offset, size_t length); int rsm_memseg_import_unmap(rsm_memseg_import_handle_t im_memseg);</pre>
DESCRIPTION	The <code>rsm_memseg_import_map()</code> and <code>rsm_memseg_import_unmap()</code> functions provide for mapping and unmapping operations on imported segments. The mapping operations are only available for native architecture interconnects such as Dolphin-SCI or Wildcat. Mapping a segment allows that segment to be accessed by CPU memory operations, saving the overhead of calling the memory access primitives described on the <code>rsm_memseg_import_get(3RSM)</code> and <code>rsm_memseg_import_put(3RSM)</code> manual pages. The <code>rsm_memseg_import_map()</code> function maps an import segment into caller's address space for the segment to be accessed by CPU memory operations. The <code>im_memseg</code> argument represents the import segment that is being mapped. The location where the process's address space is mapped to the segment is pointed to by the <code>address</code> argument. The <code>attr</code> argument can be one of the following: <code>RSM_MAP_NONE</code> The system will choose available virtual address to map and return its value in the <code>address</code> argument. <code>RSM_MAP_FIXED</code> The import segment should be mapped at the requested virtual address specified in the <code>address</code> argument. The <code>perm</code> argument determines whether read, write or a combination of accesses are permitted to the data being mapped. It can be either <code>RSM_PERM_READ</code>, <code>RSM_PERM_WRITE</code>, or <code>RSM_PERM_RDWR</code>. The <code>offset</code> argument is the byte offset location from the base of the segment being mapped to <code>address</code>. The <code>length</code> argument indicates the number of bytes from offset to be mapped. The <code>rsm_memseg_import_unmap()</code> function unmaps a previously mapped import segment.
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.
ERRORS	The <code>rsm_memseg_import_map()</code> and <code>rsm_memseg_import_unmap()</code> functions can return the following errors: <code>RSMERR_BAD_SEG_HNDL</code> Invalid segment handle.

rsm_memseg_import_map(3RSM)

The `rsm_memseg_import_map()` function can return the following errors:

RSMERR_BAD_ADDR	Invalid address.
RSMERR_BAD_LENGTH	Invalid length.
RSMERR_BAD_MEM_ALIGNMENT	The address is not aligned on a page boundary.
RSMERR_BAD_OFFSET	Invalid offset.
RSMERR_BAD_PERMS	Invalid permissions.
RSMERR_CONN_ABORTED	Connection aborted.
RSMERR_MAP_FAILED	Map failure.
RSMERR_SEG_ALREADY_MAPPED	Segment already mapped.
RSMERR_SEG_NOT_CONNECTED	Segment not connected.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_import_connect(3RSM)`, `rsm_memseg_import_get(3RSM)`, `rsm_memseg_import_put(3RSM)`, `rsm_memseg_get_pollfd(3RSM)`, `attributes(5)`

rsm_memseg_import_open_barrier(3RSM)

NAME	rsm_memseg_import_open_barrier, rsm_memseg_import_order_barrier, rsm_memseg_import_close_barrier – remote memory access error detection functions												
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_open_barrier(rsmapi_barrier_t *barrier); int rsm_memseg_import_order_barrier(rsmapi_barrier_t *barrier); int rsm_memseg_import_close_barrier(rsmapi_barrier_t *barrier);</pre>												
DESCRIPTION	The rsm_memseg_import_open_barrier() and rsm_memseg_import_close_barrier() functions provide a means of remote memory access error detection when the barrier mode is set to RSM_BARRIER_MODE_EXPLICIT. Open and close barrier operations define a span-of-time interval for error detection. A successful close barrier guarantees that remote memory access covered between the open barrier and close barrier have completed successfully. Any individual failures which may have occurred between the open barrier and close barrier occur without any notification and the failure is not reported until the close barrier. The rsm_memseg_import_order_barrier() function imposes the order-of-write completion whereby, with an order barrier, the write operations issued before the order barrier are all completed before the operations after the order barrier. Effectively, with the order barrier call, all writes within one barrier scope are ordered with respect to those in another barrier scope.												
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.												
ERRORS	The rsm_memseg_import_open_barrier(), rsm_memseg_import_order_barrier(), and rsm_memseg_import_close_barrier() functions can return the following errors: <table> <tr> <td>RSMERR_BAD_SEG_HNDL</td><td>Invalid segment handle</td></tr> <tr> <td>RSMERR_BAD_BARRIER_PTR</td><td>Invalid barrier pointer.</td></tr> </table> The rsm_memseg_close_barrier() and rsm_memseg_order_barrier() functions can return the following errors: <table> <tr> <td>RSMERR_BARRIER_UNINITIALIZED</td><td>Barrier not initialized.</td></tr> <tr> <td>RSMERR_BARRIER_NOT_OPENED</td><td>Barrier not opened.</td></tr> <tr> <td>RSMERR_BARRIER_FAILURE</td><td>Memory access error.</td></tr> <tr> <td>RSMERR_CONN_ABORTED</td><td>Connection aborted.</td></tr> </table>	RSMERR_BAD_SEG_HNDL	Invalid segment handle	RSMERR_BAD_BARRIER_PTR	Invalid barrier pointer.	RSMERR_BARRIER_UNINITIALIZED	Barrier not initialized.	RSMERR_BARRIER_NOT_OPENED	Barrier not opened.	RSMERR_BARRIER_FAILURE	Memory access error.	RSMERR_CONN_ABORTED	Connection aborted.
RSMERR_BAD_SEG_HNDL	Invalid segment handle												
RSMERR_BAD_BARRIER_PTR	Invalid barrier pointer.												
RSMERR_BARRIER_UNINITIALIZED	Barrier not initialized.												
RSMERR_BARRIER_NOT_OPENED	Barrier not opened.												
RSMERR_BARRIER_FAILURE	Memory access error.												
RSMERR_CONN_ABORTED	Connection aborted.												

`rsm_memseg_import_open_barrier(3RSM)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_memseg_import_init_barrier(3RSM)`,
`rsm_memseg_import_set_mode(3RSM)`, `attributes(5)`

rsm_memseg_import_put(3RSM)

NAME	rsm_memseg_import_put, rsm_memseg_import_put8, rsm_memseg_import_put16, rsm_memseg_import_put32, rsm_memseg_import_put64 – write to a segment
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_put(rsm_memseg_import_handle_t im_memseg, off_t offset, void *src_addr, size_t length); int rsm_memseg_import_put8(rsm_memseg_import_handle_t im_memseg, off_t offset, uint8_t datap, ulong_t rep_cnt); int rsm_memseg_import_put16(rsm_memseg_import_handle_t im_memseg, off_t offset, uint16_t datap, ulong_t rep_cnt); int rsm_memseg_import_put32(rsm_memseg_import_handle_t im_memseg, off_t offset, uint32_t datap, ulong_t rep_cnt); int rsm_memseg_import_put64(rsm_memseg_import_handle_t im_memseg, off_t offset, uint64_t datap, ulong_t rep_cnt);</pre>
DESCRIPTION	When using interconnects that allow memory mapping (see <code>rsm_memseg_import_map(3RSM)</code>), standard CPU memory operations may be used for accessing memory of a segment. If, however, a mapping is not provided, then explicitly calling these functions facilitates writing to a segment. Depending on the attributes of the extension library for the interconnect, these functions may involve doing an implicit mapping before performing the data transfer. Applications can be made interconnect-independent with respect to segment writes by using these functions. The data access error detection is performed through the use of barriers (see <code>rsm_memseg_import_open_barrier(3RSM)</code>). The default barrier operation mode is <code>RSM_BARRIER_MODE_IMPLICIT</code>, which means that around every put operation open and close barrier operations are performed automatically. Explicit error handling may also be set up for these functions (see <code>rsm_memseg_import_set_mode(3RSM)</code>). The <code>rsm_memseg_import_put()</code> function copies <i>length</i> bytes from local memory with start address <i>src_addr</i> to the imported segment <i>im_memseg</i> beginning at location <i>offset</i> from the start of the segment. The <code>rsm_memseg_import_put8()</code> function copies <i>rep_cnt</i> number of 8-bit quantities from successive local memory locations pointed to by <i>datap</i> to successive locations starting from <i>offset</i> in the imported segment. The <code>rsm_memseg_import_put16()</code> function copies <i>rep_cnt</i> number of 16-bit quantities from successive local memory locations pointed to by <i>datap</i> to successive locations starting from <i>offset</i> in the imported segment. The offset must be aligned at half-word address boundary. The <code>rsm_memseg_import_put32()</code> function copies <i>rep_cnt</i> number of 32-bit quantities from successive local memory locations pointed to by <i>datap</i> to successive locations starting from <i>offset</i> in the imported segment. The offset must be aligned at word address boundary.

rsm_memseg_import_put(3RSM)

The `rsm_memseg_import_put64()` function copies *rep_cnt* number of 64-bit quantities from successive local memory locations pointed to by *datap* to successive locations starting from *offset* in the imported segment. The offset must be aligned at double-word address boundary.

The data transfer functions that transfer small quantities of data (that is, 8-, 16-, 32-, and 64-bit quantities) perform byte swapping prior to the data transfer, in the event that the source and destination have incompatible endian characteristics.

RETURN VALUES

Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

ERRORS

These functions can return the following errors:

RSMERR_BAD_SEG_HNDL	Invalid segment handle.
RSMERR_BAD_ADDR	Bad address.
RSMERR_BAD_MEM_ALIGNMENT	Invalid memory alignment for pointer.
RSMERR_BAD_OFFSET	Invalid offset.
RSMERR_BAD_LENGTH	Invalid length.
RSMERR_PERM_DENIED	Permission denied.
RSMERR_INSUFFICIENT_RESOURCES	Insufficient resources.
RSMERR_BARRIER_UNINITIALIZED	Barrier not initialized.
RSMERR_BARRIER_FAILURE	I/O completion error.
RSMERR_CONN_ABORTED	Connection aborted.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

`rsm_memseg_import_get(3RSM)`, `rsm_memseg_import_init_barrier(3RSM)`,
`rsm_memseg_import_open_barrier(3RSM)`,
`rsm_memseg_import_set_mode(3RSM)`, `attributes(5)`

rsm_memseg_import_putv(3RSM)

NAME	rsm_memseg_import_putv, rsm_memseg_import_getv – write to a segment using a list of I/O requests
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_putv(rsm_scath_t *sg_io); int rsm_memseg_import_getv(rsm_scath_t *sg_io);</pre>
DESCRIPTION	The <code>rsm_memseg_import_putv()</code> and <code>rsm_memseg_import_getv()</code> functions provide for using a list of I/O requests rather than a single source and destination address as is done for <code>rsm_memseg_import_put(3RSM)</code> and <code>rsm_memseg_import_get(3RSM)</code> functions. The I/O vector component of the scatter-gather list (<code>sg_io</code>), allows specifying local virtual addresses or <code>local_memory_handles</code>. When a local address range is used repeatedly, it is efficient to use a handle because allocated system resources (that is, locked down local memory) are maintained until the handle is freed. The supporting functions for handles are <code>rsm_create_localmemory_handle(3RSM)</code> and <code>rsm_free_localmemory_handle(3RSM)</code>. Virtual addresses or handles may be gathered into the vector for writing to a single remote segment, or a read from a single remote segment may be scattered to the vector of virtual addresses or handles. Implicit mapping is supported for the scatter-gather type of access. The attributes of the extension library for the specific interconnect are used to determine whether mapping is necessary before any scatter-gather access. If mapping of the imported segment is a prerequisite for scatter-gather access and the mapping has not already been performed, an implicit mapping is performed for the imported segment. The I/O for the vector is then initiated. I/O for the entire vector is initiated before returning. The barrier mode attribute of the import segment determines if the I/O has completed before the function returns. A barrier mode attribute setting of <code>IMPLICIT</code> guarantees that the transfer of data is completed in the order as entered in the I/O vector. An implicit barrier open and close surrounds each list entry. If an error is detected, I/O for the vector is terminated and the function returns immediately. The residual count indicates the number of entries for which the I/O either did not complete or was not initiated. The number of entries in the I/O vector component of the scatter-gather list is specified in the <code>io_request_count</code> field of the <code>rsm_scath_t</code> pointed to by <code>sg_io</code>. The <code>io_request_count</code> is valid if greater than 0 and less than or equal to <code>RSM_MAX_SGIOREQS</code>. If <code>io_request_count</code> is not in the valid range, <code>rsm_memseg_import_putv()</code> and <code>rsm_memseg_import_getv()</code> returns <code>RSMERR_BAD_SGIO</code>. Optionally, the scatter-gather list allows support for an implicit signal post after the I/O for the entire vector has completed. This alleviates the need to do an explicit signal post after every I/O transfer operation. The means of enabling the implicit signal

rsm_memseg_import_putv(3RSM)

post involves setting the `flags` field within the scatter-gather list to `RSM_IMPLICIT_SIGPOST`. The `flags` field may also be set to `RSM_SIG_POST_NO_ACCUMULATE`, which will be passed on to the signal post operation when `RSM_IMPLICIT_SIGPOST` is set.

RETURN VALUES Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

ERRORS The `rsm_memseg_import_putv()` and `rsm_memseg_import_getv()` functions can return the following errors:

<code>RSMERR_BAD_SGIO</code>	Invalid scatter-gather structure pointer.
<code>RSMERR_BAD_SEG_HNDL</code>	Invalid segment handle.
<code>RSMERR_BAD_CTLR_HNDL</code>	Invalid controller handle.
<code>RSMERR_BAD_OFFSET</code>	Invalid offset.
<code>RSMERR_BAD_LENGTH</code>	Invalid length.
<code>RSMERR_BAD_ADDR</code>	Bad address.
<code>RSMERR_INSUFFICIENT_RESOURCES</code>	Insufficient resources.
<code>RSMERR_INTERRUPTED</code>	The operation was interrupted by a signal.
<code>RSMERR_PERM_DENIED</code>	Permission denied.
<code>RSMERR_BARRIER_FAILURE</code>	I/O completion error.
<code>RSMERR_REMOTE_NODE_UNREACHABLE</code>	Remote node not reachable.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `rsm_create_localmemory_handle(3RSM)`,
`rsm_free_localmemory_handle(3RSM)`, `attributes(5)`

rsm_memseg_import_set_mode(3RSM)

NAME	rsm_memseg_import_set_mode, rsm_memseg_import_get_mode – set or get mode for barrier scoping						
SYNOPSIS	<pre>cc [flags...] file... -lrsm [library...] #include <rsmapi.h> int rsm_memseg_import_set_mode(rsm_memseg_import_handle_t memseg, rsm_barrier_mode_t mode); int rsm_memseg_import_get_mode(rsm_memseg_import_handle_t memseg, rsm_barrier_mode_t *mode);</pre>						
DESCRIPTION	The <code>rsm_memseg_import_set_mode()</code> function provides support for optional explicit barrier scoping in the functions described on the <code>rsm_memseg_import_get(3RSM)</code> and <code>rsm_memseg_import_put(3RSM)</code> manual pages.. The two valid barrier modes are <code>RSM_BARRIER_MODE_EXPLICIT</code> and <code>RSM_BARRIER_MODE_IMPLICIT</code>. By default, the barrier mode is set to <code>RSM_BARRIER_MODE_IMPLICIT</code>. When the mode is <code>RSM_BARRIER_MODE_IMPLICIT</code>, an implicit barrier open and barrier close is applied to the put operation. Irrespective of the mode set, the barrier must be initialized using the <code>rsm_memseg_import_init_barrier(3RSM)</code> function before any barrier operations, either implicit or explicit, are used. The <code>rsm_memseg_import_get_mode()</code> function obtains the current value of the mode used for barrier scoping in put functions.						
RETURN VALUES	Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.						
ERRORS	The <code>rsm_memseg_import_set_mode()</code> and <code>rsm_memseg_import_get_mode()</code> functions can return the following errors: <code>RSMERR_BAD_SEG_HNDL</code> Invalid segment handle.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>rsm_memseg_import_get(3RSM)</code> , <code>rsm_memseg_import_init_barrier(3RSM)</code> , <code>rsm_memseg_import_put(3RSM)</code> , <code>attributes(5)</code>						

NAME	rtld_audit, la_activity, la_i86_pltenter, la_objsearch, la_objopen, la_pltexit, la_pltexit64, la_preinit, la_sparcv8_pltenter, la_sparcv9_pltenter, la_symbind32, la_symbind64, la_version – runtime linker auditing functions
SYNOPSIS	<pre> void la_activity(uintptr_t *cookie, uint_t flag); uintptr_t la_i86_pltenter(Elf32_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, La_i86_regs *regs, uint_t *flags); char *la_objsearch(const char *name, uintptr_t *cookie, uint_t flag); uint_t la_objopen(Link_map *lmp, Lmid_t lmid, uintptr_t *cookie); uintptr_t la_pltexit(Elf32_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, uintptr_t retval); uintptr_t la_pltexit64(Elf64_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, uintptr_t retval, const char *sym_name); void la_preinit(uintptr_t *cookie); uintptr_t la_sparcv8_pltenter(Elf32_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, La_sparcv8_regs *regs, uint_t *flags); uintptr_t la_sparcv9_pltenter(Elf64_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, La_sparcv8_regs *regs, uint_t *flags, const char *sym_name); uintptr_t la_symbind32(Elf32_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, uint_t *flags); uintptr_t la_symbind64(Elf64_Sym *sym, uint_t ndx, uintptr_t *refcook, uintptr_t *defcook, uint_t *flags, const char *sym_name); uint_t la_version(uint_t version); </pre>
DESCRIPTION	A runtime linker auditing library is a user-created shared object offering one or more of these interfaces that are called by the runtime linker ld.so.1(1) during process execution. See the <i>Linker and Libraries Guide</i> for a full description of the link auditing mechanism.
SEE ALSO	ld.so.1(1) <i>Linker and Libraries Guide</i>

rtld_db(3EXT)

NAME	rtld_db, rd_delete, rd_errstr, rd_event_addr, rd_event_enable, rd_event_getmsg, rd_init, rd_loadobj_iter, rd_log, rd_new, rd_objpad_enable, rd_plt_resolution, rd_reset – runtime linker debugging functions						
SYNOPSIS	<pre>cc [flag ...] file ... -lrtld_db [library ...] #include <proc_service.h> #include <rtld_db.h> void rd_delete(struct rd_agent *rdap); char *rd_errstr(rd_err_e rderr); rd_err_e rd_event_addr(rd_agent *rdap, rd_notify_t *notify); rd_err_e rd_event_enable(struct rd_agent *rdap, int onoff); rd_err_e rd_event_getmsg(struct rd_agent *rdap, rd_event_msg_t *msg); rd_err_e rd_init(int version); typedef int rl_iter_f(const rd_loadobj_t *, void *); rd_err_e rd_loadobj_iter(rd_agent_t *rap, rl_iter_f *cb, void *clnt_data); void rd_log(const int onoff); rd_agent_t *rd_new(struct ps_prochandle *php, uint_t flag); rd_err_e rd_objpad_enable(struct rd_agent *rdap, size_t padsize); rd_err_e rd_plt_resolution(rd_agent *rdap, paddr_t pc, lwpid_t lwpid, paddr_t plt_base, rd_plt_info_t *rpi); rd_err_e rd_reset(struct rd_agent *rdap);</pre>						
DESCRIPTION	The librtld_db library provides support for monitoring and manipulating runtime linking aspects of a program. There are at least two processes involved, the controlling process and one or more target processes. The controlling process is the librtld_db client that links with librtld_db and uses librtld_db to inspect or modify runtime linking aspects of one or more target processes. See the <i>Linker and Libraries Guide</i> for a full description of the runtime linker debugger interface mechanism.						
ATTRIBUTES	See attributes(5) for description of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	Safe						
SEE ALSO	ld.so.1(1), librtld_db(3LIB), libthread_db(3THR), attributes(5) <i>Linker and Libraries Guide</i>						

NAME	scalb – load exponent of a radix-independent floating-point number				
SYNOPSIS	<pre>#include <math.h> double scalb(double <i>x</i>, double <i>n</i>);</pre>				
DESCRIPTION	The <code>scalb()</code> function computes $x * r^n$, where r is the radix of the machine's floating point arithmetic. When r is 2, <code>scalb()</code> is equivalent to <code>ldexp(3C)</code> .				
RETURN VALUES	Upon successful completion, the <code>scalb()</code> function returns $x * r^n$. If the correct value would overflow, <code>scalb()</code> returns $\pm\text{HUGE\_VAL}$ (according to the sign of x) and sets <code>errno</code> to <code>ERANGE</code>. If the correct value would underflow to 0.0, <code>scalb()</code> returns 0 and sets <code>errno</code> to <code>ERANGE</code>. The <code>scalb()</code> function returns x when x is $\pm\text{Inf}$. If x or n is NaN, then <code>scalb()</code> returns NaN. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>scalb()</code> function will fail if: <code>ERANGE</code> The correct value would overflow or underflow.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>scalb()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>ldexp(3C)</code> , <code>matherr(3M)</code> , <code>attributes(5)</code>				

scalbn(3M)

NAME	scalbn – load exponent of a radix-independent floating-point number				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double scalbn(double <i>x</i>, int <i>n</i>);</pre>				
DESCRIPTION	The <code>scalbn()</code> function computes $x * r^n$, where r is the radix of the machine's floating point arithmetic.				
RETURN VALUES	Upon successful completion, the <code>scalbn()</code> function returns $x * r^n$. If the correct value would overflow, <code>scalbn()</code> returns <code>±HUGE_VAL</code> (according to the sign of x). The <code>scalbn()</code> function returns x when x is $±\text{Inf}$. If x is NaN, then <code>scalbn()</code> returns NaN.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>attributes(5)</code>				

SCF_Card_exchangeAPDU(3SMARTCARD)

NAME	SCF_Card_exchangeAPDU – send a command APDU to a card and read the card’s response										
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Card_exchangeAPDU(SCF_Card_t <i>card</i>, const uint8_t *<i>sendBuffer</i>, size_t <i>sendLength</i>, uint8_t *<i>recvBuffer</i>, size_t *<i>recvLength</i>) ;</pre>										
PARAMETERS	<table> <tr> <td><i>card</i></td><td>The card (from SCF_Terminal_getCard(3SMARTCARD)) to communicate with.</td></tr> <tr> <td><i>sendBuffer</i></td><td>A pointer to a buffer containing the command APDU.</td></tr> <tr> <td><i>sendLength</i></td><td>The number of bytes in the <i>sendBuffer</i> (that is, the size of the command APDU).</td></tr> <tr> <td><i>recvBuffer</i></td><td>A pointer to a buffer in which the card’s reply APDU should be stored. This buffer can be the same as the <i>sendBuffer</i> to allow the application to conserve memory usage. The buffer must be large enough to store the expected reply.</td></tr> <tr> <td><i>recvLength</i></td><td>The caller specifies the maximum size of the <i>recvBuffer</i> in <i>recvLength</i>. The library uses this value to prevent overflowing the buffer. When the reply is received, the library sets <i>recvLength</i> to the actual size of the reply APDU that was stored in the <i>recvBuffer</i>.</td></tr> </table>	<i>card</i>	The card (from SCF_Terminal_getCard(3SMARTCARD)) to communicate with.	<i>sendBuffer</i>	A pointer to a buffer containing the command APDU.	<i>sendLength</i>	The number of bytes in the <i>sendBuffer</i> (that is, the size of the command APDU).	<i>recvBuffer</i>	A pointer to a buffer in which the card’s reply APDU should be stored. This buffer can be the same as the <i>sendBuffer</i> to allow the application to conserve memory usage. The buffer must be large enough to store the expected reply.	<i>recvLength</i>	The caller specifies the maximum size of the <i>recvBuffer</i> in <i>recvLength</i> . The library uses this value to prevent overflowing the buffer. When the reply is received, the library sets <i>recvLength</i> to the actual size of the reply APDU that was stored in the <i>recvBuffer</i> .
<i>card</i>	The card (from SCF_Terminal_getCard(3SMARTCARD)) to communicate with.										
<i>sendBuffer</i>	A pointer to a buffer containing the command APDU.										
<i>sendLength</i>	The number of bytes in the <i>sendBuffer</i> (that is, the size of the command APDU).										
<i>recvBuffer</i>	A pointer to a buffer in which the card’s reply APDU should be stored. This buffer can be the same as the <i>sendBuffer</i> to allow the application to conserve memory usage. The buffer must be large enough to store the expected reply.										
<i>recvLength</i>	The caller specifies the maximum size of the <i>recvBuffer</i> in <i>recvLength</i> . The library uses this value to prevent overflowing the buffer. When the reply is received, the library sets <i>recvLength</i> to the actual size of the reply APDU that was stored in the <i>recvBuffer</i> .										
DESCRIPTION	The SCF_Card_exchangeAPDU() function sends a binary command to the card and reads the reply. The application is responsible for constructing a valid command and providing a receive buffer large enough to hold the reply. Generally, the command and reply will be ISO7816-formatted APDUs (Application Protocol Data Units), but the SCF library does not examine or verify the contents of the buffers. If the caller needs to perform a multi-step transaction that must not be interrupted, SCF_Card_lock(3SMARTCARD) should be used to prevent other applications from communicating with the card during the transaction. Similarly, calls to SCF_Card_exchangeAPDU() must be prepared to retry the call if SCF_STATUS_CARDLOCKED is returned. An ISO7816-formatted command APDU always begins with a mandatory 4 byte header (CLA, INS, P1, and P2), followed by a variable length body (zero or more bytes). For details on the APDUs supported by a specific card, consult the documentation provided by the card manufacturer or applet vendor. An ISO7816-formatted reply APDU consists of zero or more bytes of data, followed by a mandatory 2 byte status trailer (SW1 and SW2).										
RETURN VALUES	If the APDU is successfully sent and a reply APDU is successfully read, SCF_STATUS_SUCCESS is returned with <i>recvBuffer</i> and <i>recvLength</i> set appropriately. Otherwise, an error value is returned and both <i>recvBuffer</i> and <i>recvLength</i> remain unaltered.										

SCF_Card_exchangeAPDU(3SMARTCARD)

ERRORS

The SCF_Card_exchangeAPDU() function will fail if:

SCF_STATUS_BADARGS	Neither <i>sendBuffer</i> , <i>recvBuffer</i> , nor <i>recvLength</i> can be null pointers. The value of <i>recvLength</i> must be at least 2.
SCF_STATUS_BADHANDLE	The card has been closed or is invalid.
SCF_STATUS_CARDLOCKED	The APDU cannot be sent because the card is locked by another application.
SCF_STATUS_CARDREMOVED	The card object cannot be used because the card represented by the SCF_Card_t has been removed
SCF_STATUS_COMMERROR	The connection to the server was closed.
SCF_STATUS_FAILED	An internal error occurred.
SCF_STATUS_NOSPACE	The specified size of <i>recvBuffer</i> is too small to hold the complete reply APDU.

EXAMPLES

EXAMPLE 1 Send a command to the card.

```
SCF_Status_t status;
SCF_Card_t myCard;
uint8_t commandAPDU[] = {0x00, 0xa4, 0x00, 0x00, 0x02, 0x3f, 0x00};
uint8_t replyAPDU[256];
uint32_t commandSize = sizeof(commandAPDU);
uint32_t replySize = sizeof(replyAPDU);
/* (...call SCF_Terminal_getCard to open myCard...) */

/* Send the ISO7816 command to select the card's MF. */
status = SCF_Card_exchangeAPDU(myCard, commandAPDU, commandSize,
    replyAPDU, &replySize);
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("Received a %d byte reply.\n", replySize);
printf("SW1=0x%02.2x SW2=0x%02.2x\n",
    replyAPDU[replySize-2], replyAPDU[replySize-1]);

/* ... */
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

libsmartcard(3LIB), SCF_Card_lock(3SMARTCARD),
SCF_Terminal_getCard(3SMARTCARD), attributes(5)

SCF_Card_lock(3SMARTCARD)

NAME	SCF_Card_lock, SCF_Card_unlock – perform mutex locking on a card
SYNOPSIS	<pre>cc [flag...] file... -lsmartcard [library...] #include <smartcard/scf.h> SCF_Status_t SCF_Card_lock(SCF_Card_t card, unsigned int timeout); SCF_Status_t SCF_Card_unlock(SCF_Card_t card);</pre>
PARAMETERS	<i>card</i> The card (from SCF_Terminal_getCard(3SMARTCARD)) to be locked. <i>timeout</i> The maximum number of seconds SCF_Card_lock() should wait for a card locked by another application to become unlocked. A value of 0 results in SCF_Card_lock() returning immediately if a lock cannot be immediately obtained. A value of SCF_TIMEOUT_MAX results in SCF_Card_lock() waiting forever to obtain a lock.
DESCRIPTION	Locking a card allows an application to perform a multi-APDU transaction (that is, multiple calls to SCF_Card_exchangeAPDU(3SMARTCARD)) without interference from other smartcard applications. The lock is enforced by the server, so that other applications that attempt to call SCF_Card_exchangeAPDU() or SCF_Card_reset(3SMARTCARD) will be denied access to the card. Applications should restrict use of locks only to brief critical sections. Otherwise it becomes difficult for multiple applications to share the same card. When a lock is granted to a specific SCF_Card_t card object, only that object can be used to access the card and subsequently release the lock. If a misbehaving application holds a lock for an extended period, the lock can be broken by having the user remove and reinsert the smartcard. It is an error to attempt to lock a card when the caller already holds a lock on the card (that is, calling SCF_Card_lock() twice in a succession). Unlocking a card that is not locked (or was already unlocked) can be performed without causing an error. An application might find that it is unable to lock the card, or communicate with it because SCF_Card_exchangeAPDU() keeps returning SCF_STATUS_CARDLOCKED. If this situation persists, it might indicate that another application has not released its lock on the card. The user is able to forcibly break a lock by removing the card and reinserting it, after which the application must call SCF_Terminal_getCard(3SMARTCARD) to access the "new" card. In this situation an application should retry for a reasonable period of time, and then alert the user that the operation could not be completed because the card is in use by another application and that removing or reinserting the card will resolve the problem.
RETURN VALUES	If the card is successfully locked or unlocked, SCF_STATUS_SUCCESS is returned. Otherwise, the lock status of the card remains unchanged and an error value is returned.
ERRORS	The SCF_Card_lock() and SCF_Card_unlock() functions will fail if:

SCF_Card_lock(3SMARTCARD)

SCF_STATUS_BADHANDLE	The specified card has been closed or is invalid.
SCF_STATUS_CARDLOCKED	There is a lock present on the card, but it is not held by the specified card object. For example, the caller is attempting to unlock a card locked by another application.
SCF_STATUS_CARDREMOVED	The card object cannot be used because the card represented by the SCF_Card_t has been removed.
SCF_STATUS_COMMERROR	The connection to the server was lost.
SCF_STATUS_DOUBLELOCK	The caller has already locked this card and is attempting to lock it again.
SCF_STATUS_FAILED	An internal error occurred.
SCF_STATUS_TIMEOUT	The <i>timeout</i> expired before the call was able to obtain the lock.

EXAMPLES

EXAMPLE 1 Use a card lock.

```
SCF_Status_t status;
SCF_Card_t myCard;

/* (...call SCF_Terminal_getCard to open myCard...) */

status = SCF_Card_lock(myCard, 15);
if (status == SCF_STATUS_TIMEOUT) {
    printf("Unable to get a card lock, someone else has a lock.\n");
    exit(0);
}
else if (status != SCF_STATUS_SUCCESS) exit(1);

/* Send the first APDU */
SCF_Card_exchangeAPDU(myCard, ...);

/* Send the second APDU */
SCF_Card_exchangeAPDU(myCard, ...);

status = SCF_Card_unlock(myCard);

/* ... */
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SCF_Card_lock(3SMARTCARD)

SEE ALSO libsmartcard(3LIB), SCF_Card_exchangeAPDU(3SMARTCARD),
SCF_Card_reset(3SMARTCARD), SCF_Terminal_getCard(3SMARTCARD),
attributes(5)

SCF_Card_reset(3SMARTCARD)

NAME	SCF_Card_reset – perform a reset of a smartcard										
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Card_reset(SCF_Card_t <i>card</i>) ;</pre>										
PARAMETERS	<i>card</i> The card (from SCF_Terminal_getCard(3SMARTCARD)) to be reset										
DESCRIPTION	The SCF_Card_reset () function causes the specified smartcard to be reset by the terminal. A card can be reset only if it has not been locked (with SCF_Card_lock(3SMARTCARD)) by another client. A client wishing to reset a card should either first call SCF_Card_lock () to obtain the card lock, or be prepared to retry the reset operation if it fails because another client holds the card lock. When the card is reset, any SCF_Card_t object representing the card will continue to remain valid after the reset. When the reset occurs, an SCF_EVENT_CARDRESET event will be sent to all registered event listeners for the terminal (assuming they registered for this event). This is the only notification of a reset provided to SCF clients. When a client receives this event, it should be prepared to reinitialize any state on the card that might have been interrupted by the reset. New information about the card (for example, ATR, if it changed) can also be available from SCF_Card_getInfo(3SMARTCARD).										
RETURN VALUES	If the card is successfully reset, SCF_STATUS_SUCCESS is returned. Otherwise, the status of the card remains unchanged and an error value is returned.										
ERRORS	The SCF_Card_reset () function will fail if: <table> <tr> <td>SCF_STATUS_BADHANDLE</td><td>The specified card has been closed or is invalid.</td></tr> <tr> <td>SCF_STATUS_CARDLOCKED</td><td>The card cannot be reset because another client holds a lock on the card.</td></tr> <tr> <td>SCF_STATUS_CARDREMOVED</td><td>The card cannot be reset because the card represented by the SCF_Card_t has been removed.</td></tr> <tr> <td>SCF_STATUS_COMMERROR</td><td>The connection to the server was lost.</td></tr> <tr> <td>SCF_STATUS_FAILED</td><td>An internal error occurred.</td></tr> </table>	SCF_STATUS_BADHANDLE	The specified card has been closed or is invalid.	SCF_STATUS_CARDLOCKED	The card cannot be reset because another client holds a lock on the card.	SCF_STATUS_CARDREMOVED	The card cannot be reset because the card represented by the SCF_Card_t has been removed.	SCF_STATUS_COMMERROR	The connection to the server was lost.	SCF_STATUS_FAILED	An internal error occurred.
SCF_STATUS_BADHANDLE	The specified card has been closed or is invalid.										
SCF_STATUS_CARDLOCKED	The card cannot be reset because another client holds a lock on the card.										
SCF_STATUS_CARDREMOVED	The card cannot be reset because the card represented by the SCF_Card_t has been removed.										
SCF_STATUS_COMMERROR	The connection to the server was lost.										
SCF_STATUS_FAILED	An internal error occurred.										
EXAMPLES	EXAMPLE 1 Reset a card. <pre>SCF_Status_t status; SCF_Card_t myCard; /* (...call SCF_Terminal_getCard to open myCard...) */ status = SCF_Card_lock(myCard, SCF_TIMEOUT_MAX); if (status != SCF_STATUS_SUCCESS) exit(1);</pre>										

EXAMPLE 1 Reset a card. *(Continued)*

```
status = SCF_Card_reset(myCard);
if (status != SCF_STATUS_SUCCESS) exit(1);

status = SCF_Card_unlock(myCard);
if (status != SCF_STATUS_SUCCESS) exit(1);

/* ... */
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Card_getInfo(3SMARTCARD),
SCF_Card_lock(3SMARTCARD),
SCF_Terminal_addEventListener(3SMARTCARD),
SCF_Terminal_getCard(3SMARTCARD), attributes(5)

SCF_Session_close(3SMARTCARD)

NAME	SCF_Session_close, SCF_Terminal_close, SCF_Card_close – close a smartcard session, terminal, or card
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Session_close(SCF_Session_t <i>session</i>) ; SCF_Status_t SCF_Terminal_close(SCF_Terminal_t <i>terminal</i>) ; SCF_Status_t SCF_Card_close(SCF_Card_t <i>card</i>) ;</pre>
PARAMETERS	<i>card</i> An object that was returned from SCF_Terminal_getCard(3SMARTCARD) <i>session</i> An object that was returned from SCF_Session_getSession(3SMARTCARD) <i>terminal</i> An object that was returned from SCF_Session_getTerminal(3SMARTCARD)
DESCRIPTION	These functions release the resources (memory, threads, and others) that were allocated within the library when the session, terminal, or card was opened. Any storage allocated by calls to SCF_Session_getInfo(3SMARTCARD), SCF_Terminal_getInfo(3SMARTCARD), or SCF_Card_getInfo(3SMARTCARD) is deallocated when the associated object is closed. Attempts to access results from these interfaces after the object has been closed results in undefined behavior. If a card that was locked by SCF_Card_lock(3SMARTCARD) is closed, the lock is automatically released. When a terminal is closed, any event listeners on that terminal object are removed and any cards that were obtained with the terminal are closed. Similarly, closing a session will close any terminals or cards obtained with that session. These are the only cases where the library will automatically perform a close. Once closed, a session, terminal, or card object can no longer be used by an SCF function. Any attempt to do so results in an SCF_STATUS_BADHANDLE error. The sole exception is that closing an object, even if already closed, is always a successful operation.
RETURN VALUES	Closing a handle is always a successful operation that returns SCF_STATUS_SUCCESS. The library can safely detect handles that are invalid or already closed.
EXAMPLES	EXAMPLE 1 Close each object explicitly. <pre>SCF_Status_t status; SCF_Session_t mySession; SCF_Terminal_t myTerminal; SCF_Card_t myCard; status = SCF_Session_getSession(&mySession); if (status != SCF_STATUS_SUCCESS) exit(1); status = SCF_Session_getTerminal(mySession, NULL, &myTerminal); if (status != SCF_STATUS_SUCCESS) exit(1);</pre>

SCF_Session_close(3SMARTCARD)

EXAMPLE 1 Close each object explicitly. (Continued)

```
status = SCF_Terminal_getCard(myTerminal, &myCard);
if (status != SCF_STATUS_SUCCESS) exit(1);

/* (Do interesting things with smartcard...) */

SCF_Card_close(myCard);
SCF_Terminal_close(myTerminal);
SCF_Session_close(mySession);
```

EXAMPLE 2 Allow the library to close objects.

```
SCF_Status_t status;
SCF_Session_t mySession;
SCF_Terminal_t myTerminal;
SCF_Card_t myCard;

status = SCF_Session_getSession(&mySession);
if (status != SCF_STATUS_SUCCESS) exit(1);
status = SCF_Session_getTerminal(mySession, NULL, &myTerminal);
if (status != SCF_STATUS_SUCCESS) exit(1);
status = SCF_Terminal_getCard(myTerminal, &myCard);
if (status != SCF_STATUS_SUCCESS) exit(1);

/* (Do interesting things with smartcard...) */

SCF_Session_close(mySession);
/* myTerminal and myCard have been closed by the library. */
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Card_getInfo(3SMARTCARD), SCF_Card_lock(3SMARTCARD), SCF_Session_getInfo(3SMARTCARD), SCF_Session_getSession(3SMARTCARD), SCF_Session_getTerminal(3SMARTCARD), SCF_Terminal_getCard(3SMARTCARD), SCF_Terminal_getInfo(3SMARTCARD), attributes(5)

SCF_Session_freeInfo(3SMARTCARD)

NAME	SCF_Session_freeInfo, SCF_Terminal_freeInfo, SCF_Card_freeInfo – deallocate information storage
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Session_freeInfo(SCF_Session_t <i>session</i>, void *<i>value</i>) ; SCF_Status_t SCF_Terminal_freeInfo(SCF_Terminal_t <i>terminal</i>, void *<i>value</i>) ; SCF_Status_t SCF_Card_freeInfo(SCF_Card_t <i>card</i>, void *<i>value</i>) ;</pre>
PARAMETERS	<i>card</i> An object that was returned from SCF_Terminal_getCard(3SMARTCARD). This object must be associated with the information value being freed. <i>session</i> An object that was returned from SCF_Session_getSession(3SMARTCARD). This object must be associated with the information value being freed. <i>terminal</i> An object that was returned from SCF_Session_getTerminal(3SMARTCARD). This object must be associated with the information value being freed. <i>value</i> A pointer that was returned from a call to SCF_Session_getInfo(3SMARTCARD), SCF_Terminal_getInfo(3SMARTCARD), or SCF_Card_getInfo(3SMARTCARD).
DESCRIPTION	When information is requested for an object (for example, by using SCF_Session_getInfo()), the result is placed in memory allocated for that request. This memory must eventually be deallocated, or a memory leak will result. The deallocation of memory can occur in one of two ways. ■ The simplest method is to allow the smartcard library to automatically deallocate memory when the object associated with the information is closed. For example, when SCF_Card_close(3SMARTCARD) is called, any information obtained from SCF_Card_getInfo() for that card object is deallocated. The application is not required to call SCF_Card_freeInfo() at all. ■ If the object persists for a long period of time, the application can explicitly request the information to be deallocated without closing the object, so that memory is not wasted on unneeded storage. Similarly, if an application repeatedly requests information about an object (even the same information), the application can explicitly request deallocation as needed, so that memory usage does not continue to increase until the object is closed. In general, requesting information to be deallocated can be used to reduce runtime memory bloat. Attempts to access deallocated memory result in undefined behavior.

SCF_Session_freeInfo(3SMARTCARD)

RETURN VALUES If the information is successfully deallocated, SCF_STATUS_SUCCESS is returned. Otherwise, an error value is returned.

ERRORS These functions will fail if:

SCF_STATUS_BADARGS	The specified value cannot be deallocated, possibly because of an invalid pointer, a value already deallocated, or because the value is not associated with the specified session, terminal, or card.
SCF_STATUS_BADHANDLE	The specified session, terminal, or card has been closed or is invalid.
SCF_STATUS_FAILED	An internal error occurred.

EXAMPLES **EXAMPLE 1** Free information.

```
char *terminalName;
SCF_Status_t status;
SCF_Terminal_t myTerminal;

/* (...call SCF_Session_getTerminal to open myTerminal...) */

status = SCF_Terminal_getInfo(myTerminal, "name", &terminalName);
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("The terminal name is %s\n", terminalName);

status = SCF_Terminal_freeInfo(myTerminal, terminalName);
if (status != SCF_STATUS_SUCCESS) exit(1);
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Session_getInfo(3SMARTCARD), SCF_Session_getSession(3SMARTCARD), SCF_Session_getTerminal(3SMARTCARD), SCF_Terminal_getCard(3SMARTCARD), attributes(5)

SCF_Session_getInfo(3SMARTCARD)

NAME	SCF_Session_getInfo, SCF_Terminal_getInfo, SCF_Card_getInfo – retrieve information about a session, terminal, or card
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Session_getInfo(SCF_Session_t <i>session</i>, const char *<i>name</i>, void *<i>value</i>); SCF_Status_t SCF_Terminal_getInfo(SCF_Terminal_t <i>terminal</i>, const char *<i>name</i>, void *<i>value</i>); SCF_Status_t SCF_Card_getInfo(SCF_Card_t <i>card</i>, const char *<i>name</i>, void *<i>value</i>);</pre>
PARAMETERS	<i>card</i> An object that was returned from SCF_Terminal_getCard(3SMARTCARD). <i>name</i> The name of a property for which a value is to be returned. The name is case-sensitive. <i>session</i> An object that was returned from SCF_Session_getSession(3SMARTCARD). <i>terminal</i> An object that was returned from SCF_Session_getTerminal(3SMARTCARD). <i>value</i> The value of the property. The actual type of the value depends on what property was being queried.
DESCRIPTION	These functions obtain information about a session, terminal, or card. The information returned represents the current state of the object and can change between calls. Each call allocates new storage for the returned result. This storage is tracked internally and is deallocated when the object is closed. An application repeatedly asking for information can cause memory bloat until the object is closed. The application can optionally call SCF_Session_freeInfo(3SMARTCARD), SCF_Terminal_freeInfo(3SMARTCARD), or SCF_Card_freeInfo(3SMARTCARD) to cause immediate deallocation of the value. Applications must not use other means such as free(3C) to deallocate the memory. Applications must not access values that have been deallocated. For example, accessing a Card's ATR after the card has been closed results in undefined behavior. For a session, the valid property names and value types are: <i>terminalnames</i> (pointer to char **) The list of terminal names that can currently be used in this session. The returned value is an array of char *, each element of the list is a pointer to a terminal name. The end of the array is denoted by a null pointer. The first element of the list is the default terminal for the session, which will be used when SCF_Session_getTerminal() is called with a null pointer for the terminal name.

SCF_Session_getInfo(3SMARTCARD)

For a terminal, the standard property names and value types are as follows. Some terminal drivers can define additional driver-specific properties.

name (pointer to char *)

The name of the terminal. If the default terminal was used (a null pointer was passed to `SCF_Session_getTerminal()`), the value will contain the actual name of the default terminal. For example, "MyInternalCardReader".

type (pointer to char *)

The type of the terminal. For example, "SunISCR1".

devname (pointer to char *)

Information about how the device is attached to the system. This can be a UNIX device name (for example, "/dev/scmi2c0") or some other terminal-specific string describing its relation to the system.

For a card, the valid property names and value types are:

type (pointer to char *)

The type of the smartcard, as recognized by the framework (For example, "Cyberflex"). If the framework does not recognize the card type, "UnknownCard" is returned.

atr (pointer to struct `SCF_BinaryData_t` *)

The Answer To Reset (ATR) data returned by the card when it was last inserted or reset. The structure member `length` denotes how many bytes are in the ATR. The structure member `data` is a pointer to the actual ATR bytes.

RETURN VALUES

Upon success, `SCF_STATUS_SUCCESS` is returned and *value* will contain the requested information. Otherwise, an error value is returned and *value* remains unaltered.

ERRORS

These functions will fail if:

`SCF_STATUS_BADARGS`

Either *name* or *value* is a null pointer.

`SCF_STATUS_BADHANDLE`

The session, terminal, or card has been closed or is invalid.

`SCF_STATUS_FAILED`

An internal error occurred.

`SCF_STATUS_UNKNOWNPROPERTY`

The property specified by *name* was not found.

EXAMPLES

EXAMPLE 1 Simple string information.

```
SCF_Status_t status;
SCF_Terminal_t myTerminal;
const char *myName, *myType;

/* (...call SCF_Session_getTerminal to open myTerminal...) */

status = SCF_Terminal_getInfo(myTerminal, "name", &myName);
if (status != SCF_STATUS_SUCCESS) exit(1);
status = SCF_Terminal_getInfo(myTerminal, "type", &myType);
```

SCF_Session_getInfo(3SMARTCARD)

EXAMPLE 1 Simple string information. *(Continued)*

```
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("The terminal called %s is a %s\n", myName, myType);
```

EXAMPLE 2 Display the names of all terminals available in the session.

```
SCF_Status_t status;
SCF_Session_t mySession;
const char **myList; /* Technically "const char * const *". */
int i;

/* (...call SCF_Session_getSession to open mySession...) */

status = SCF_Session_getInfo(mySession, "terminalnames", &myList);
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("The following terminals are available:\n");
for (i=0; myList[i] != NULL; i++) {
    printf("%d: %s\n", i, myList[i]);
}
```

EXAMPLE 3 Display the card's ATR.

```
SCF_Status_t status;
SCF_Card_t myCard;
struct SCF_BinaryData_t *myATR;
int i;

/* (...call SCF_Terminal_getCard to open myCard...) */

status = SCF_Card_getInfo(myCard, "atr", &myATR);
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("The card's ATR is: 0x");
for(i=0; i < myATR->length; i++) {
    printf("%02.2x", myATR->data[i]);
}
printf("\n");
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SCF_Session_getInfo(3SMARTCARD)

SEE ALSO libsmartcard(3LIB), SCF_Session_freeInfo(3SMARTCARD),
SCF_Session_getSession(3SMARTCARD),
SCF_Session_getTerminal(3SMARTCARD),
SCF_Terminal_getCard(3SMARTCARD), attributes(5)

SCF_Session_getSession(3SMARTCARD)

NAME	SCF_Session_getSession – establish a context with a system’s smartcard framework						
SYNOPSIS	<pre>cc [flag...] file... -lsmartcard [library...] #include <smartcard/scf.h> SCF_Status_t SCF_Session_getSession(SCF_Session_t *session);</pre>						
PARAMETERS	<i>session</i> A pointer to an SCF_Session_t. If a session is successfully established, the session will be returned through this parameter.						
DESCRIPTION	The SCF_Session_getSession() function establishes a session with the Solaris Smart Card Framework (SCF). Once a session has been opened, the session can be used with SCF_Session_getTerminal(3SMARTCARD) to access a smartcard terminal (reader). Information about the session can be obtained by calling SCF_Session_getInfo(3SMARTCARD). When the session is no longer needed, SCF_Session_close(3SMARTCARD) should be called to end the session and release session resources. Closing a session will also close any terminals and cards opened within the session. An application usually needs to open only a single session. For example, multiple terminals can be opened from the same session. If an application opens additional sessions, each call will return independent (different) sessions.						
RETURN VALUES	Upon success, SCF_STATUS_SUCCESS is returned and <i>session</i> contains a valid session. If a session could not be established, an error value is returned and <i>session</i> remains unaltered.						
ERRORS	The SCF_Session_getSession() function will fail if: <table> <tr> <td>SCF_STATUS_BADARGS</td><td>The <i>session</i> argument is a null pointer.</td></tr> <tr> <td>SCF_STATUS_COMMERROR</td><td>The library was unable to contact the smartcard server daemon (ocfserv(1M)), or the library was unable to obtain a session from the server.</td></tr> <tr> <td>SCF_STATUS_FAILED</td><td>An internal error occurred.</td></tr> </table>	SCF_STATUS_BADARGS	The <i>session</i> argument is a null pointer.	SCF_STATUS_COMMERROR	The library was unable to contact the smartcard server daemon (ocfserv(1M)), or the library was unable to obtain a session from the server.	SCF_STATUS_FAILED	An internal error occurred.
SCF_STATUS_BADARGS	The <i>session</i> argument is a null pointer.						
SCF_STATUS_COMMERROR	The library was unable to contact the smartcard server daemon (ocfserv(1M)), or the library was unable to obtain a session from the server.						
SCF_STATUS_FAILED	An internal error occurred.						
EXAMPLES	EXAMPLE 1 Establish a session with the framework. <pre>SCF_Status_t status; SCF_Session_t mySession; status = SCF_Session_getSession(&mySession); if (status != SCF_STATUS_SUCCESS) exit(1); /* Proceed with other smartcard operations. */</pre>						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						

SCF_Session_getSession(3SMARTCARD)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Session_close(3SMARTCARD),
SCF_Session_getInfo(3SMARTCARD),
SCF_Session_getTerminal(3SMARTCARD), attributes(5)

SCF_Session_getTerminal(3SMARTCARD)

NAME	SCF_Session_getTerminal – establish a context with a smartcard terminal (reader)						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Session_getTerminal(SCF_Session_t <i>session</i>, const char *<i>terminalName</i>, SCF_Terminal_t *<i>terminal</i>);</pre>						
PARAMETERS	<table> <tr> <td><i>session</i></td><td>The session (from SCF_Session_getSession(3SMARTCARD)) containing a terminal to be opened.</td></tr> <tr> <td><i>terminal</i></td><td>A pointer to an SCF_Terminal_t. If the terminal is successfully opened, a handle for the terminal will be returned through this parameter.</td></tr> <tr> <td><i>terminalName</i></td><td>Specifies the name of the terminal to access. If <i>terminalName</i> is a null pointer, it indicates that the library should connect with the default terminal for the session.</td></tr> </table>	<i>session</i>	The session (from SCF_Session_getSession(3SMARTCARD)) containing a terminal to be opened.	<i>terminal</i>	A pointer to an SCF_Terminal_t. If the terminal is successfully opened, a handle for the terminal will be returned through this parameter.	<i>terminalName</i>	Specifies the name of the terminal to access. If <i>terminalName</i> is a null pointer, it indicates that the library should connect with the default terminal for the session.
<i>session</i>	The session (from SCF_Session_getSession(3SMARTCARD)) containing a terminal to be opened.						
<i>terminal</i>	A pointer to an SCF_Terminal_t. If the terminal is successfully opened, a handle for the terminal will be returned through this parameter.						
<i>terminalName</i>	Specifies the name of the terminal to access. If <i>terminalName</i> is a null pointer, it indicates that the library should connect with the default terminal for the session.						
DESCRIPTION	The SCF_Session_getTerminal() function establishes a context with a specific smartcard terminal (also known as a reader) in the session. Terminal objects are used for detecting card movement (insertion or removal) and to create card objects for accessing a specific card. The list of available terminal names can be retrieved by calling SCF_Session_getInfo(3SMARTCARD). Unless the user explicitly requests a specific terminal, applications should use the session's default terminal by calling SCF_Session_getTerminal() with a null pointer for the terminal name. This eliminates the need to first process an available-terminal list with just one element on systems with only a single smartcard terminal. On multi-terminal systems, the user can preconfigure one of the terminals as the default (or preferred) terminal. See USAGE below. If SCF_Session_getTerminal() is called multiple times in the same session to access the same physical terminal, the same SCF_Terminal_t will be returned in each call. Multithreaded applications must take care to avoid having one thread close a terminal that is still needed by another thread. This can be accomplished by coordination within the application or by having each thread open a separate session to avoid interference. When the terminal is no longer needed, SCF_Terminal_close(3SMARTCARD) should be called to release terminal resources. Closing a terminal will also close any cards opened from the terminal.						
RETURN VALUES	Upon success, SCF_STATUS_SUCCESS is returned and <i>terminal</i> contains the opened terminal. Otherwise, an error value is returned and <i>terminal</i> remains unaltered.						
ERRORS	The SCF_Session_getTerminal() function will fail if: <table> <tr> <td>SCF_STATUS_BADARGS</td><td>The <i>terminal</i> argument is a null pointer.</td></tr> <tr> <td>SCF_STATUS_BADHANDLE</td><td>The session was closed or is invalid.</td></tr> </table>	SCF_STATUS_BADARGS	The <i>terminal</i> argument is a null pointer.	SCF_STATUS_BADHANDLE	The session was closed or is invalid.		
SCF_STATUS_BADARGS	The <i>terminal</i> argument is a null pointer.						
SCF_STATUS_BADHANDLE	The session was closed or is invalid.						

SCF_Session_getTerminal(3SMARTCARD)

SCF_STATUS_BADTERMINAL	The specified <i>terminalName</i> is not valid for this session, or the default terminal could not be opened because there are no terminals available in this session.
SCF_STATUS_COMMERROR	The connection to the server was lost.
SCF_STATUS_FAILED	An internal error occurred.

EXAMPLES

EXAMPLE 1 Use the default terminal.

```
SCF_Status_t status;
SCF_Session_t mySession;
SCF_Terminal_t myTerminal;
char *myName;

/* (...call SCF_Session_getSession to open mySession...) */

status = SCF_Session_getTerminal(mySession, NULL, &myTerminal);
if (status != SCF_STATUS_SUCCESS) exit(1);

status = SCF_Terminal_getInfo(myTerminal, "name", &myName);
if (status != SCF_STATUS_SUCCESS) exit(1);

printf("Please insert a card into the terminal named %s\n", myName);

/* ... */
```

EXAMPLE 2 Open a terminal by name.

```
SCF_Status_t status;
SCF_Session_t mySession;
SCF_Terminal_t myTerminal;
char *myName;

/* (...call SCF_Session_getSession to open mySession...) */

/*
 * The name should be selected from the list of terminal names
 * available from SCF_Session_getInfo, but it could also be
 * read from an application's config file or from user input.
 */
myName = "SunInternalReader";

status = SCF_Session_getTerminal(mySession, myName, &myTerminal);
if (status == SCF_STATUS_BADTERMINAL) {
    printf("There is no terminal named %s.\n", myName);
    exit(1);
} else if (status != SCF_STATUS_SUCCESS) exit(2);

/* ... */
```

USAGE

When using the Solaris OCF smartcard framework, the default reader is specified by the `ocf.client.default.defaultreader` property. If this property is not set, the first available reader is chosen as the default. Users can set the

SCF_Session_getTerminal(3SMARTCARD)

SCF_DEFAULT_TERMINAL environment variable to the name of a terminal to override the normal default. The smartcard utility can also be used to add terminals to or remove terminals from the system. See smartcard(1M) for information on how to add or modify the OCF property.

Terminals can be accessed only by the user who expected to have physical access to the terminal. By default, this user is assumed to be the owner of /dev/console and the superuser. Certain terminals such as Sun Ray appliances can use a different method to restrict access to the terminal.

The framework also uses the DISPLAY environment variable to further restrict which terminals are listed for a user. By default, terminals are associated with the ":0" display. Sun Ray terminals are associated with the display for that session, for example ":25". If the DISPLAY environment variable is not set or is a display on another host, it is treated as though it were set to ":0". Terminals not associated with the user's DISPLAY are not listed. To override this behaviour, the SCF_FILTER_KEY environment variable can be set to the desired display, for example ":0", ":25", and so on. To list all terminals to which a user has access, SCF_FILTER_KEY can be set to the special value of ".*".

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO smartcard(1M), libsmartcard(3LIB), SCF_Session_getInfo(3SMARTCARD), SCF_Session_getSession(3SMARTCARD), SCF_Terminal_close(3SMARTCARD), attributes(5)

NAME	SCF_strerror – get a string describing a status code						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> const char *SCF_strerror(SCF_Status_t <i>error</i>);</pre>						
PARAMETERS	<i>error</i> A value returned from a smartcard SCF function call. A list of all current codes is contained in <smartcard/scf.h>						
DESCRIPTION	The SCF_strerror() function provides a mechanism for generating a brief message that describes each SCF_Status_t error code. An application might use the message when displaying or logging errors. The string returned by the function does not contain any newline characters. Returned strings must not be modified or freed by the caller.						
RETURN VALUES	A pointer to a valid string is always returned. If the provided <i>error</i> is not a valid SCF error code, a string is returned stating that the error code is unknown. A null pointer is never returned.						
EXAMPLES	EXAMPLE 1 Report a fatal error. <pre>SCF_Status_t status; SCF_Session_t mySession; status = SCF_Session_getSession(&mySession); if (status != SCF_STATUS_SUCCESS) { printf("Smartcard startup error: %s\n", SCF_strerror(status)); exit(1); } /* ... */</pre>						
USAGE	Messages returned from SCF_strerror() are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C). The C locale is used if the native strings could not be loaded.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Interface Stability</td><td>Evolving</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	libsmartcard(3LIB), SCF_Session_getSession(3SMARTCARD), strerror(3C), attributes(5)						

SCF_Terminal_addEventListener(3SMARTCARD)

NAME	SCF_Terminal_addEventListener, SCF_Terminal_updateEventListener, SCF_Terminal_removeEventListener – receive asynchronous event notification	
SYNOPSIS	<pre>cc [flag...] file... -lsmartcard [library...] #include <smartcard/scf.h> SCF_Status_t SCF_Terminal_addEventListener(SCF_Terminal_t terminal, SCF_Event_t events, void(*callback)(SCF_Event_t, SCF_Terminal_t, void *), void *userData, SCF_ListenerHandle_t *listenerHandle); SCF_Status_t SCF_Terminal_updateEventListener(SCF_Terminal_t terminal, SCF_ListenerHandle_t listenerHandle, SCF_Event_t events); SCF_Status_t SCF_Terminal_removeEventListener(SCF_Terminal_t terminal, SCF_ListenerHandle_t listenerHandle);</pre>	
PARAMETERS	<i>terminal</i>	A terminal (from SCF_Session_getTerminal(3SMARTCARD)) to which the event listener should be added or removed.
	<i>events</i>	Events to deliver to the callback. An event will not be delivered if it is not listed. The caller can register for multiple events by performing a bitwise OR of the desired events. The valid events are: SCF_EVENT_ALL All of the events listed below will be delivered. SCF_EVENT_CARDINSERTED A smartcard was inserted into the terminal. SCF_EVENT_CARDREMOVED A smartcard was removed from the terminal. SCF_EVENT_CARDPRESENT Indicates that a card was present in the terminal when the event listener was first added. This event allows event listeners to determine the initial state of the terminal before an insert or remove event occurs. Either this event or the SCF_EVENT_CARDABSENT (see below) event will be delivered only once upon adding an event listener and immediately before any other events are delivered. Future card movements will generate SCF_EVENT_CARDINSERTED and SCF_EVENT_CARDREMOVED events, but not SCF_EVENT_CARDPRESENT or SCF_EVENT_CARDABSENT events. An event listener can assume that if a SCF_EVENT_CARDPRESENT event is delivered, the next card movement event will be a SCF_EVENT_CARDREMOVED. SCF_EVENT_CARDABSENT Indicates that a card was not present in the terminal when the event listener was first added. This event allows event listeners to determine the initial state of the terminal before an insert or

SCF_Terminal_addEventListener(3SMARTCARD)

remove event occurs. Either this event or the SCF_EVENT_CARDPRESENT event (see above) will be delivered only once upon adding an event listener and immediately before any other events are delivered. Future card movements will generate SCF_EVENT_CARDINSERTED and SCF_EVENT_CARDREMOVED events, but not SCF_EVENT_CARDPRESENT or SCF_EVENT_CARDABSENT events. An event listener can assume that if a SCF_EVENT_CARDABSENT event is delivered, the next card movement event will be a SCF_EVENT_CARDINSERTED.

SCF_EVENT_CARDRESET

The smartcard currently present has been reset (see SCF_Card_reset(3SMARTCARD)).

SCF_EVENT_TERMINALCLOSED

The terminal is in the process of being closed (due to a call to SCF_Session_close(3SMARTCARD) or SCF_Terminal_close(3SMARTCARD)), so no further events will be delivered. The *terminal* argument provided to the callback will still be valid.

SCF_EVENT_COMMERROR

The connection to the server has been lost. No further events will be delivered.

<i>callback</i>	A function pointer that will be executed when the desired event occurs. The function must take three arguments. The first is a SCF_Event_t containing the event that occurred. The second argument is an SCF_Terminal_t containing the terminal on which the event occurred. The third is a void * that can be used to provide arbitrary data to the <i>callback</i> when it is executed.
<i>userData</i>	A pointer to arbitrary user data. The data is not accessed by the library. The pointer is simply provided to the callback when an event is issued. This argument can safely be set to NULL if not needed. The callback must be able to handle this case.
<i>listenerHandle</i>	A unique "key" that is provided by SCF_Terminal_addEventListener() to refer to a specific event listener registration. This allows multiple event listeners to be selectively updated or removed.

DESCRIPTION These functions allow an application to receive notification of events on a terminal as they occur. The concept is similar to a signal handler. When an event occurs, a thread in the SCF library will execute the provided *callback* function. Once added, the listener will receive events until it is removed or either the terminal or session is closed.

When the callback function is executed, the callback arguments specify the event that occurred and the terminal on which it occurred. Additionally, each callback will receive the *userData* pointer that was provided when the listener was added. The library does

SCF_Terminal_addEventListener(3SMARTCARD)

not make a copy of the memory pointed to by *userData*, so applications must take care not to deallocate that memory until it is known that the callback will no longer access it (for example, by removing the event listener). Each invocation of the callback will be for exactly one event. If the library needs to deliver multiple events, they will be dispatched one at a time. Because the callback is executed from a thread, any operations it performs must be thread safe. For each callback registration, the library creates a new thread to deliver events to that callback. The callback is expected to perform minimal work and return quickly.

An application can add multiple callbacks on a terminal. Any event that occurs will be delivered to all listeners that registered for that event type. The same callback can be registered multiple times. Each call to `SCF_Terminal_addEventListener()` will result in a new `SCF_ListenerHandle_t`. The events a callback receives can be changed by calling `SCF_Session_updateEventListener()` with the handle that was returned when the listener was initially added. If the listener is set to receive no events (that is, the events parameter has no bits set), the listener will remain registered but will not receive any events. To remove a listener and release allocated resources, use `SCF_Terminal_removeEventListener()` or close the terminal.

RETURN VALUES

If the event listener was successfully added or removed, `SCF_STATUS_SUCCESS` is returned. Otherwise, an error value is returned and the internal list of registered event listeners remains unaltered.

ERRORS

These functions will fail if:

<code>SCF_STATUS_BADARGS</code>	The callback function pointer and/or <i>listenerHandle</i> is null, or an unknown event was specified.
<code>SCF_STATUS_BADHANDLE</code>	The specified terminal has been closed or is invalid, or the event listener handle could not be found to update or remove.
<code>SCF_STATUS_COMMERROR</code>	The connection to the server was lost.
<code>SCF_STATUS_FAILED</code>	An internal error occurred.

EXAMPLES

EXAMPLE 1 Register for card movements.

```
struct myState_t {
    int isStateKnown;
    int isCardPresent;
};

void myCallback(SCF_Event_t event, SCF_Terminal_t eventTerminal,
void *data) {
    struct myState_t *state = data;
    if (event == SCF_EVENT_CARDINSERTED) {
        printf("--- Card inserted ---\n");
        state->isCardPresent = 1;
    }
    else if (event == SCF_EVENT_CARDREMOVED) {
        printf("--- Card removed ---\n");
        state->isCardPresent = 0;
    }
}
```

SCF_Terminal_addEventListener(3SMARTCARD)

EXAMPLE 1 Register for card movements. (Continued)

```

    }
    state->isStateKnown = 1;
}

main() {
    SCF_Status_t status;
    SCF_Terminal_t myTerminal;
    SCF_ListenerHandle_t myListener;
    struct myState_t myState;

    /* (...call SCF_Session_getTerminal to open myTerminal...) */

    myState.isStateKnown = 0;
    status = SCF_Terminal_addEventListener(myTerminal,
        SCF_EVENT_CARDINSERTED|SCF_EVENT_CARDREMOVED, &myCallback,
        &myState, &myListener);
    if (status != SCF_STATUS_SUCCESS) exit(1);

    while(1) {
        if (!myState.isStateKnown)
            printf("Waiting for first event...\n");
        else {
            if (myState.isCardPresent)
                printf("Card is present.\n");
            else
                printf("Card is not present.\n");
        }
        sleep(1);
    }
}

```

EXAMPLE 2 Use different callbacks for each event.

```

void myInsertCallback(SCF_Event_t event, SCF_Terminal_t eventTerminal,
    void *data) {

    /* ... */
}

void myRemoveCallback(SCF_Event_t event, SCF_Terminal_t eventTerminal,
    void *data) {
    /* ... */
}

main () {
    SCF_Status_t status;
    SCF_Terminal_t terminal;
    SCF_ListenerHandle_t myListener1, myListener2, myListener3;
    int foo, bar;

    /* (...call SCF_Session_getTerminal to open myTerminal...) */

    status = SCF_Terminal_addEventListener(myTerminal,
        SCF_EVENT_CARDINSERTED, &myInsertCallback, &foo,

```

SCF_Terminal_addEventListener(3SMARTCARD)

EXAMPLE 2 Use different callbacks for each event. *(Continued)*

```

        &myListener1);
    if (status != SCF_STATUS_SUCCESS) exit(1);

    status = SCF_Terminal_addEventListener(myTerminal,
        SCF_EVENT_CARDREMOVED, &myRemoveCallback, &foo,
        &myListener2);
    if (status != SCF_STATUS_SUCCESS) exit(1);

    status = SCF_Terminal_addEventListener(myTerminal,
        SCF_EVENT_CARDREMOVED, &myRemoveCallback, &bar,
        &myListener3);
    if (status != SCF_STATUS_SUCCESS) exit(1);

    /*
     * At this point, when each insertion occurs, myInsertCallback
     * will be called once (with a pointer to foo). When each removal
     * occurs, myRemoveCallback will be called twice. One call will
     * be given a pointer to foo, and the other will be given a
     * pointer to bar.
     */

    status = SCF_Terminal_removeEventListener(myTerminal,
        myListener2);
    if (status != SCF_STATUS_SUCCESS) exit(1);

    /*
     * Now, when a removal occurs, myRemoveCallback will only be
     * called once, with a pointer to bar.
     */

    /* ... */
}

```

EXAMPLE 3 Use initial state events to show user the terminal state in a GUI.

```

void myCallback(SCF_Event_t event, SCF_Terminal_t eventTerminal,
    void *unused) {
    if (event == SCF_EVENT_CARDPRESENT) {
        /* Set initial icon to a terminal with a card present. */
    }
    else if (event == SCF_EVENT_CARDABSENT) {
        /* Set initial icon to a terminal without a card present. */
    }
    else if (event == SCF_EVENT_CARDINSERTED) {
        /* Show animation for card being inserted into a terminal. */
    }
    else if (event == SCF_EVENT_CARDREMOVED) {
        /* Show animation for card being removed from a terminal. */
    }
}

main() {
    SCF_Terminal_t myTerminal;
    SCF_ListenerHandle_t myListener;
}

```

SCF_Terminal_addEventListener(3SMARTCARD)

EXAMPLE 3 Use initial state events to show user the terminal state in a GUI. *(Continued)*

```
/* (...call SCF_Session_getTerminal to open myTerminal...) */

status = SCF_Terminal_addEventListener(myTerminal,
    SCF_EVENT_ALL, &myCallback, NULL, &myListener);
if (status != SCF_STATUS_SUCCESS) exit(1);

/* ... */
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Card_reset(3SMARTCARD), SCF_Session_close(3SMARTCARD), SCF_Session_getTerminal(3SMARTCARD), SCF_Session_updateEventListener(3SMARTCARD), SCF_Terminal_close(3SMARTCARD), SCF_Terminal_removeEventListener(3SMARTCARD), attributes(5)

SCF_Terminal_getCard(3SMARTCARD)

NAME	SCF_Terminal_getCard – establish a context with a smartcard								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Terminal_getCard(SCF_Terminal_t <i>terminal</i>, SCF_Card_t *<i>card</i>) ;</pre>								
PARAMETERS	<i>card</i> A pointer to a SCF_Card_t. If the smartcard is successfully opened, a handle for the card will be returned through this parameter. <i>terminal</i> The terminal (from SCF_Session_getTerminal(3SMARTCARD)) containing a smartcard to open.								
DESCRIPTION	The SCF_Terminal_getCard() function establishes a context with a specific smartcard in a terminal. Card objects can be used to send APDUs (Application Protocol Data Units) to the card with SCF_Card_exchangeAPDU(3SMARTCARD). When the card is no longer needed, SCF_Card_close(3SMARTCARD) should be called to release allocated resources. If SCF_Terminal_getCard() is called multiple times in the same session to access the same physical card (while the card remains inserted), the same SCF_Card_t will be returned in each call. The library cannot identify specific cards, so when a card is reinserted it will be represented by a new SCF_Card_t. Multithreaded applications must take care to avoid having one thread close a card that is still needed by another thread. This can be accomplished by coordination within the application, or by having each thread open a separate session to avoid interference.								
RETURN VALUES	If a working card is present in the reader, SCF_STATUS_SUCCESS is returned and <i>card</i> is a valid reference to the card. Otherwise, an error value is returned and <i>card</i> remains unaltered.								
ERRORS	The SCF_Terminal_getCard() function will fail if: <table> <tr> <td>SCF_STATUS_BADARGS</td><td>The <i>card</i> argument is a null pointer.</td></tr> <tr> <td>SCF_STATUS_BADHANDLE</td><td>The specified terminal has been closed or is invalid.</td></tr> <tr> <td>SCF_STATUS_FAILED</td><td>An internal error occurred.</td></tr> <tr> <td>SCF_STATUS_NOCARD</td><td>No card is present in the terminal.</td></tr> </table>	SCF_STATUS_BADARGS	The <i>card</i> argument is a null pointer.	SCF_STATUS_BADHANDLE	The specified terminal has been closed or is invalid.	SCF_STATUS_FAILED	An internal error occurred.	SCF_STATUS_NOCARD	No card is present in the terminal.
SCF_STATUS_BADARGS	The <i>card</i> argument is a null pointer.								
SCF_STATUS_BADHANDLE	The specified terminal has been closed or is invalid.								
SCF_STATUS_FAILED	An internal error occurred.								
SCF_STATUS_NOCARD	No card is present in the terminal.								
EXAMPLES	EXAMPLE 1 Access a smartcard. <pre>SCF_Status_t status; SCF_Terminal_t myTerminal; SCF_Card_t myCard; /* (...call SCF_Session_getTerminal to open myTerminal...) */ status = SCF_Terminal_getCard(myTerminal, &myCard); if (status == SCF_STATUS_NOCARD) {</pre>								

SCF_Terminal_getCard(3SMARTCARD)

EXAMPLE 1 Access a smartcard. (Continued)

```
        printf("Please insert your smartcard and try again.\n");
        exit(0);
    }
    else if (status != SCF_STATUS_SUCCESS) exit(1);

    /* (...go on to use the card with SCF_Card_exchangeAPDU()) */
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO libsmartcard(3LIB), SCF_Card_close(3SMARTCARD), SCF_Card_exchangeAPDU(3SMARTCARD), SCF_Card_getInfo(3SMARTCARD), SCF_Card_lock(3SMARTCARD), SCF_Session_getTerminal(3SMARTCARD), attributes(5)

SCF_Terminal_waitForCardPresent(3SMARTCARD)

NAME	SCF_Terminal_waitForCardPresent, SCF_Terminal_waitForCardAbsent, SCF_Card_waitForCardRemoved – wait for a card to be inserted or removed			
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsmartcard [<i>library...</i>] #include <smartcard/scf.h> SCF_Status_t SCF_Terminal_waitForCardPresent(SCF_Terminal_t <i>terminal</i>, unsigned int <i>timeout</i>); SCF_Status_t SCF_Terminal_waitForCardAbsent(SCF_Terminal_t <i>terminal</i>, unsigned int <i>timeout</i>); SCF_Status_t SCF_Card_waitForCardRemoved(SCF_Card_t <i>card</i>, unsigned int <i>timeout</i>);</pre>			
PARAMETERS	<i>card</i>	A card that was returned from SCF_Terminal_getCard(3SMARTCARD).		
	<i>terminal</i>	A terminal that was returned from SCF_Session_getTerminal(3SMARTCARD).		
	<i>timeout</i>	The maximum number of seconds to wait for the desired state to be reached. If the timeout is 0, the function will immediately return SCF_STATUS_TIMEOUT if the terminal or card is not in the desired state. A timeout of SCF_TIMEOUT_MAX can be specified to indicate that the function should never timeout.		
DESCRIPTION	These functions determine if a card is currently available in the specified terminal. The SCF_Card_waitForCardRemoved() function differs from SCF_Terminal_waitForCardAbsent() in that it checks to see if a specific card has been removed. If another card (or even the same card) has since been reinserted, SCF_Card_waitForCardRemoved() will report that the old card was removed, while the SCF_Terminal_waitForCardAbsent() will instead report that there is a card present. If the desired state is already true, the function will immediately return SCF_STATUS_SUCCESS. Otherwise it will wait for a change to the desired state, or for the timeout to expire, whichever occurs first. Unlike an event listener (SCF_Terminal_addEventListener(3SMARTCARD)), these functions return the state of the terminal, not just events. To use an electronics analogy, event listeners are edge-triggered, while these functions are level-triggered.			
RETURN VALUES	If the desired state is reached before the timeout expires, SCF_STATUS_SUCCESS is returned. If the timeout expires, SCF_STATUS_TIMEOUT is returned. Otherwise, an error value is returned.			
ERRORS	These functions will fail if: <table><tr><td>SCF_STATUS_BADHANDLE</td><td>The specified <i>terminal</i> or <i>card</i> has been closed or is invalid.</td></tr></table>		SCF_STATUS_BADHANDLE	The specified <i>terminal</i> or <i>card</i> has been closed or is invalid.
SCF_STATUS_BADHANDLE	The specified <i>terminal</i> or <i>card</i> has been closed or is invalid.			

SCF_Terminal_waitForCardPresent(3SMARTCARD)

SCF_STATUS_COMMERROR The server closed the connection.

SCF_STATUS_FAILED An internal error occurred.

EXAMPLES

EXAMPLE 1 Determine if a card is currently inserted.

```
int isCardCurrentlyPresent(SCF_Terminal_t myTerminal) {
    SCF_Status_t status;

    /*
     * The timeout of zero makes sure this call will always
     * return immediately.
     */
    status = SCF_Terminal_waitForCardPresent(myTerminal, 0);

    if (status == SCF_STATUS_SUCCESS) return (TRUE);
    else if (status == SCF_STATUS_TIMEOUT) return (FALSE);

    /*
     * For other errors, this example just assumes no card
     * is present. We don't really know.
     */
    return (FALSE);
}
```

EXAMPLE 2 Remind the user every 5 seconds to remove their card.

```
SCF_Status_t status;
SCF_Terminal_t myTerminal;

/* (...call SCF_Session_getTerminal to open myTerminal...) */

status = SCF_Terminal_waitForCardAbsent(myTerminal, 0);
while (status == SCF_STATUS_TIMEOUT) {
    printf("Please remove the card from the terminal!\n");
    status = SCF_Terminal_waitForCardAbsent(myTerminal, 5);
}

if (status == SCF_STATUS_SUCCESS)
    printf("Thank you.\n");
else
    exit(1);

/* ... */
```

EXAMPLE 3 Demonstrate the difference between the card-specific and terminal-specific calls.

```
SCF_Status_t status;
SCF_Terminal_t myTerminal;
SCF_Card_t myCard;

/* (...call SCF_Session_getTerminal to open myTerminal...) */

status = SCF_Terminal_getCard(myTerminal, &myCard);
if (status != SCF_STATUS_SUCCESS) exit(1);
```

SCF_Terminal_waitForCardPresent(3SMARTCARD)

EXAMPLE 3 Demonstrate the difference between the card-specific and terminal-specific calls. *(Continued)*

```
/*
 * While we sleep, assume user removes the card
 * and inserts another card.
 */
sleep(10);

status = SCF_Terminal_waitForCardAbsent(myTerminal, 0);
/*
 * In this case, status is expected to be SCF_STATUS_TIMEOUT, as there
 * is a card present.
 */

status = SCF_Card_waitForCardRemoved(myCard, 0);
/*
 * In this case, status is expected to be SCF_STATUS_SUCCESS, as the
 * card returned from SCF_Terminal_getCard was indeed removed (even
 * though another card is currently in the terminal).
 */

/* ... */
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

libsmartcard(3LIB), SCF_Session_getTerminal(3SMARTCARD),
SCF_Terminal_addEventListener(3SMARTCARD),
SCF_Terminal_getCard(3SMARTCARD), attributes(5)

NAME	sendfile – send files over sockets or copy files to files																
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -lsendfile [<i>library</i>...] #include <sys/sendfile.h> ssize_t sendfile(int <i>out_fd</i>, int <i>in_fd</i>, off_t *<i>off</i>, size_t <i>len</i>);</pre>																
DESCRIPTION	The <code>sendfile()</code> function copies data from <i>out_fd</i> to <i>in_fd</i> starting at offset <i>off</i> and of length <i>len</i> bytes. The <i>in_fd</i> argument should be a file descriptor to a regular file opened for reading. See <code>open(2)</code>. The <i>out_fd</i> argument should be a file descriptor to a regular file opened for writing or to a connected AF_INET or AF_INET6 socket of SOCK_STREAM type. See <code>socket(3SOCKET)</code>. The <i>off</i> argument is a pointer to a variable holding the input file pointer position from which the data will be read. After <code>sendfile()</code> has completed, the variable will be set to the offset of the byte following the last byte that was read. The <code>sendfile()</code> function does not modify the current file pointer of <i>in_fd</i>, but does modify the file pointer for <i>out_fd</i> if it is a regular file. The <code>sendfile()</code> function can also be used to send buffers by pointing <i>in_fd</i> to SFV_FD_SELF.																
RETURN VALUES	Upon successful completion, <code>sendfile()</code> returns the total number of bytes written to <i>out_fd</i> and also updates the offset to point to the byte that follows the last byte read. Otherwise, it returns -1, and <code>errno</code> is set to indicate an error.																
ERRORS	The <code>sendfile()</code> function will fail if: <table> <tr> <td>EAFNOSUPPORT</td><td>The implementation does not support the specified address family for socket.</td></tr> <tr> <td>EAGAIN</td><td>Mandatory file or record locking is set on either the file descriptor or output file descriptor if it points at regular files. O_NDELAY or O_NONBLOCK is set, and there is a blocking record lock. An attempt has been made to write to a stream that cannot accept data with the O_NDELAY or the O_NONBLOCK flag set.</td></tr> <tr> <td>EBADF</td><td>The <i>out_fd</i> or <i>in_fd</i> argument is either not a valid file descriptor, <i>out_fd</i> is not opened for writing, or <i>in_fd</i> is not opened for reading.</td></tr> <tr> <td>EINVAL</td><td>The offset cannot be represented by the <code>off_t</code> structure, or the length is negative when cast to <code>ssize_t</code>.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while accessing the file system.</td></tr> <tr> <td>ENOTCONN</td><td>The socket is not connected.</td></tr> <tr> <td>EOPNOTSUPP</td><td>The socket type is not supported.</td></tr> <tr> <td>EPIPE</td><td>The <i>out_fd</i> argument is no longer connected to the peer endpoint.</td></tr> </table>	EAFNOSUPPORT	The implementation does not support the specified address family for socket.	EAGAIN	Mandatory file or record locking is set on either the file descriptor or output file descriptor if it points at regular files. O_NDELAY or O_NONBLOCK is set, and there is a blocking record lock. An attempt has been made to write to a stream that cannot accept data with the O_NDELAY or the O_NONBLOCK flag set.	EBADF	The <i>out_fd</i> or <i>in_fd</i> argument is either not a valid file descriptor, <i>out_fd</i> is not opened for writing, or <i>in_fd</i> is not opened for reading.	EINVAL	The offset cannot be represented by the <code>off_t</code> structure, or the length is negative when cast to <code>ssize_t</code> .	EIO	An I/O error occurred while accessing the file system.	ENOTCONN	The socket is not connected.	EOPNOTSUPP	The socket type is not supported.	EPIPE	The <i>out_fd</i> argument is no longer connected to the peer endpoint.
EAFNOSUPPORT	The implementation does not support the specified address family for socket.																
EAGAIN	Mandatory file or record locking is set on either the file descriptor or output file descriptor if it points at regular files. O_NDELAY or O_NONBLOCK is set, and there is a blocking record lock. An attempt has been made to write to a stream that cannot accept data with the O_NDELAY or the O_NONBLOCK flag set.																
EBADF	The <i>out_fd</i> or <i>in_fd</i> argument is either not a valid file descriptor, <i>out_fd</i> is not opened for writing, or <i>in_fd</i> is not opened for reading.																
EINVAL	The offset cannot be represented by the <code>off_t</code> structure, or the length is negative when cast to <code>ssize_t</code> .																
EIO	An I/O error occurred while accessing the file system.																
ENOTCONN	The socket is not connected.																
EOPNOTSUPP	The socket type is not supported.																
EPIPE	The <i>out_fd</i> argument is no longer connected to the peer endpoint.																
USAGE	The <code>sendfile()</code> function has a transitional interface for 64-bit file offsets. See <code>lf64(5)</code> .																

sendfile(3EXT)

EXAMPLES

EXAMPLE 1 Sending a Buffer Over a Socket

The following example demonstrates how to send the buffer *buf* over a socket. At the end, it prints the number of bytes transferred over the socket from the buffer. It assumes that *addr* will be filled up appropriately, depending upon where to send the buffer.

```
int tfd;
off_t baddr;
struct sockaddr_in sin;
char buf[64 * 1024];
in_addr_t addr;

tfd = socket(AF_INET, SOCK_STREAM, 0);
if (tfd == -1) {
    perror("socket");
    exit(1);
}

sin.sin_family = AF_INET;
sin.sin_addr = addr; /* Fill in the appropriate address. */
sin.sin_port = htons(2345);
if (connect(tfd, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("connect");
    exit(1);
}

baddr = (off_t)buf;
len = sendfile(tfd, SFV_FD_SELF, &baddr, len);

if (len == -1) {
    perror("sendfile");
    exit(1);
}

printf("Transferred %d bytes from buffer to socket\n", len);
```

EXAMPLE 2 Transferring Files to Sockets

The following program demonstrates a transfer of files to sockets:

```
int ffd, tfd;
off_t off;
struct sockaddr_in sin;
in_addr_t addr;
int len;
struct stat stat_buf;

ffd = open("file", O_RDONLY);
if (ffd == -1) {
    perror("open");
    exit(1);
}

tfd = socket(AF_INET, SOCK_STREAM, 0);
if (tfd == -1) {
```

EXAMPLE 2 Transferring Files to Sockets (Continued)

```
        perror("socket");
        exit(1);
    }

    sin.sin_family = AF_INET;
    sin.sin_addr = addr; /* Fill in the appropriate address. */
    sin.sin_port = htons(2345);
    if (connect(tfd, (struct sockaddr *) &sin, sizeof(sin)) < 0) {
        perror("connect");
        exit(1);
    }

    if (fstat(ffd, &stat_buf) == -1) {
        perror("fstat");
        exit(1);
    }

    len = sendfile(tfd, ffd, &off, stat_buf.st_size);

    if (len == -1) {
        perror("sendfile");
        exit(1);
    }
}
```

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

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

FILES /usr/lib/libsendfile.so.1 shared object file

SEE ALSO open(2), socket(3SOCKET), attributes(5), lf64(5)

sendfilev(3EXT)

NAME	sendfilev – send a file								
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lsendfile [<i>library...</i>] #include <sys/sendfile.h> ssize_t sendfilev(int <i>fildev</i>, const struct sendfilevec *<i>vec</i>, int <i>sfvcnt</i>, size_t *<i>xferred</i>);</pre>								
DESCRIPTION	The <code>sendfilev()</code> function attempts to write data from the <code>sfvcnt</code> buffers specified by the members of <code>vec</code> array: <code>vec[0]</code>, <code>vec[1]</code>, ..., <code>vec[sfvcnt-1]</code>. <i>fildev</i> is a file descriptor to a regular file or to a <code>AF_NCA</code>, <code>AF_INET</code>, or <code>AF_INET6</code> family type <code>SOCK_STREAM</code> socket that is open for writing. This function is analogous to the <code>writenv()</code> system call. See <code>writenv(2)</code>. However, instead of sending out chunks of data, <code>sendfilev()</code> can read input data from data buffers or file descriptors. The following is the <code>sendfilevec</code> structure: <pre>typedef struct sendfilevec { int sfv_fd; /* input fd */ uint_t sfv_flag; /* Flags. see below */ off_t sfv_off; /* offset to start reading from */ size_t sfv_len; /* amount of data */ } sendfilevec_t; #define SFV_FD_SELF (-2)</pre> To send a file, open the file for reading. Point <code>sfv_fd</code> to the file descriptor returned as a result. See <code>open(2)</code>. <code>sfv_off</code> should contain the offset within the file. <code>sfv_len</code> should have the length of the file to be transferred. The <i>xferred</i> parameter is updated to record the total number of bytes written to <code>out_fd</code>. The <code>sfv_flag</code> field is reserved and should be set to zero. To send data directly from the address space of the process, set <code>sfv_fd</code> to <code>SFV_FD_SELF</code>. <code>sfv_off</code> should point to the data, with <code>sfv_len</code> containing the length of the buffer.								
PARAMETERS	The <code>sendfilev()</code> function supports the following parameters: <table> <tr> <td><i>fildev</i></td><td>A file descriptor to a regular file or to a <code>AF_NCA</code>, <code>AF_INET</code>, or <code>AF_INET6</code> family type <code>SOCK_STREAM</code> socket that is open for writing. For <code>AF_NCA</code>, the protocol type should be zero.</td></tr> <tr> <td><i>vec</i></td><td>An array of <code>SENDFILEVEC_T</code>, as defined in the <code>sendfilevec</code> structure above.</td></tr> <tr> <td><i>sfvcnt</i></td><td>The number of members in <i>vec</i>.</td></tr> <tr> <td><i>xferred</i></td><td>The total number of bytes written to <code>out_fd</code>.</td></tr> </table>	<i>fildev</i>	A file descriptor to a regular file or to a <code>AF_NCA</code> , <code>AF_INET</code> , or <code>AF_INET6</code> family type <code>SOCK_STREAM</code> socket that is open for writing. For <code>AF_NCA</code> , the protocol type should be zero.	<i>vec</i>	An array of <code>SENDFILEVEC_T</code> , as defined in the <code>sendfilevec</code> structure above.	<i>sfvcnt</i>	The number of members in <i>vec</i> .	<i>xferred</i>	The total number of bytes written to <code>out_fd</code> .
<i>fildev</i>	A file descriptor to a regular file or to a <code>AF_NCA</code> , <code>AF_INET</code> , or <code>AF_INET6</code> family type <code>SOCK_STREAM</code> socket that is open for writing. For <code>AF_NCA</code> , the protocol type should be zero.								
<i>vec</i>	An array of <code>SENDFILEVEC_T</code> , as defined in the <code>sendfilevec</code> structure above.								
<i>sfvcnt</i>	The number of members in <i>vec</i> .								
<i>xferred</i>	The total number of bytes written to <code>out_fd</code> .								

RETURN VALUES	Upon successful completion, <code>sendfilev()</code> returns total number of bytes written to <code>out_fd</code> . Otherwise, it returns -1, and <code>errno</code> is set to indicate an error. <i>xferred</i> contains the amount of data successfully transferred, which can be used to discover the error vector.	
ERRORS	<code>EAFNOSUPPORT</code>	The implementation does not support the specified address family for socket.
	<code>EPROTOTYPE</code>	The socket type is not supported.
	<code>EBADF</code>	The <i>fildev</i> argument is not a valid descriptor open for writing or an <code>sfv_fd</code> is invalid or not open for reading.
	<code>EACCES</code>	The process does not have appropriate privileges or one of the files pointed by <code>sfv_fd</code> does not have appropriate permissions.
	<code>EPIPE</code>	The <i>fildev</i> argument is a socket that has been shut down for writing.
	<code>EIO</code>	An I/O error occurred while accessing the file system.
	<code>EFAULT</code>	The <i>vec</i> argument points to an illegal address.
	<code>EFAULT</code>	The <i>xferred</i> argument points to an illegal address.
	<code>EINVAL</code>	The <i>sfvcnt</i> argument was less than or equal to 0. One of the <code>sfv_len</code> in <i>vec</i> array was less than or equal to 0, or greater than the file size. An <code>sfv_fd</code> is not seekable.
	<code>EAGAIN</code>	Mandatory file or record locking is set on either the file descriptor or output file descriptor if it points at regular files. <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock. An attempt has been made to write to a stream that cannot accept data with the <code>O_NDELAY</code> or the <code>O_NONBLOCK</code> flag set.
USAGE	The <code>sendfilev()</code> function has a transitional interface for 64-bit file offsets. See <code>lf64(5)</code> .	
EXAMPLES	The following example sends 2 vectors, one of HEADER data and a file of length 100 over <code>sockfd</code>. <code>sockfd</code> is in a connected state, that is, <code>socket()</code>, <code>accept()</code>, and <code>bind()</code> operation are complete. <pre>#include <sys/sendfile.h> . . . int main (int argc, char eargv[]){ int sockfd; ssize_t ret; size_t xfer;</pre>	

sendfilev(3EXT)

```
struct sendfilevec vec[2];
.
.
.
vec[0].sfv_fd = SFV_FD_SELF;
vec[0].sfv_flag = 0;
vec[0].sfv_off = "HEADER_DATA";
vec[0].sfv_len = strlen("HEADER_DATA");
vec[1].sfv_fd = open("input_file",.... );
vec[1].sfv_flag = 0;
vec[1].sfv_off = 0;
vec[1].sfv_len = 100;

ret = sendfilev(sockfd, vec, 2, &xfer);
.
.
.
}
```

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

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

SEE ALSO open(2), writev(2), attributes(5)

NAME	setproject – place process in new project with attendant resource controls, resource pools, and attributes						
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i>... -lproject [<i>library</i> ...] #include <project.h> int setproject(const char *<i>project_name</i>, const char *<i>user_name</i>, uint_t <i>flags</i>);</pre>						
DESCRIPTION	The setproject () function provides a simplified method for the association of a user process with a project and its various resource management attributes, as stored in the project(4) name service database. These attributes include resource control settings, resource pool membership, and third party attributes (which are ignored by setproject ().) If <i>user_name</i> is a valid member of the project specified by <i>project_name</i>, as determined by inproj(3PROJECT), setproject () will create a new task with settaskid(2) using task flags specified by <i>flags</i>, use setrctl(2) to associate various resource controls with the process, task, and project, and bind the calling process to the appropriate resource pool with pool_set_binding(3POOL). Resource controls not explicitly specified in the project entry will be preserved. If <i>user_name</i> is a name of the superuser (user with UID equal to 0), the setproject () function skips the inproj(3PROJECT) check described above and allows the superuser to join any project. The current process will not be bound to a resource pool if the resource pools facility (see pooladm(1M)) is inactive. The setproject () function will succeed whether or not the project specified by <i>project_name</i> specifies a project.pool attribute. If the resource pools facility is active, setproject () will fail if the project does not specify a project.pool attribute and there is no designated pool accepting default assignments. The setproject () function will also fail if there is a specified project.pool attribute for a nonexistent pool.						
RETURN VALUES	Upon successful completion, setproject () returns 0. If any of the resource control assignments failed but the project assignment, pool binding, and task creation succeeded, an integer value corresponding to the offset into the key-value pair list of the failed attribute assignment is returned. If the project assignment or task creation was not successful, setproject () returns SETPROJ_ERR_TASK and sets errno to indicate the error. In the event of a pool binding failure, setproject () returns SETPROJ_ERR_POOL and sets errno to indicate the error. Additional error information can be retrieved from pool_error(3POOL).						
ERRORS	The setproject () function will fail during project assignment or task creation if: <table> <tr> <td>EACCES</td><td>The invoking task was created with the TASK_FINAL flag.</td></tr> <tr> <td>EINVAL</td><td>The project ID associated with the given project is not within the range of valid project IDs, invalid flags were specified, or <i>user_name</i> is NULL.</td></tr> <tr> <td>EPERM</td><td>The effective user of the calling process is not superuser.</td></tr> </table>	EACCES	The invoking task was created with the TASK_FINAL flag.	EINVAL	The project ID associated with the given project is not within the range of valid project IDs, invalid flags were specified, or <i>user_name</i> is NULL.	EPERM	The effective user of the calling process is not superuser.
EACCES	The invoking task was created with the TASK_FINAL flag.						
EINVAL	The project ID associated with the given project is not within the range of valid project IDs, invalid flags were specified, or <i>user_name</i> is NULL.						
EPERM	The effective user of the calling process is not superuser.						

setproject(3PROJECT)

ESRCH The specified user is not a valid user of the given project, *user_name* is not valid user name, or *project_name* is not valid project name.

The `setproject()` function will fail during pool binding if:

EACCES No resource pool accepting default bindings exists.

EPERM The effective user of the calling process is not superuser.

ESRCH The specified resource pool is unknown

If `setproject()` returns an offset into the key-value pair list, the returned error value is associated with `setrctl(2)` for resource control attributes.

USAGE The `setproject()` function recognizes a name-structured value pair for the attributes in the `project(4)` database with the following format:

```
entity.control=(privilege,value,action,action,...),...
```

where *privilege* is one of BASIC or PRIVILEGED, *value* is a numeric value with optional units, and *action* is one of none, deny, and `signal=signum` or `signal=SIGNAME`. For instance, to set a series of progressively more assertive control values on a project's per-process CPU time, specify

```
process.max-cpu-time=(PRIVILEGED,1000s,signal=SIGXRES), \
(PRIVILEGED,1250, signal=SIGTERM),(PRIVILEGED,1500,signal=SIGKILL)
```

To prevent a task from exceeding a total of 128 LWPs, specify a resource control with

```
task.max-lwps=(PRIVILEGED,128,deny)
```

The project attribute, `project.pool`, specifies the pool to which processes associated with the project entry should be bound. Its format is:

```
project.pool=pool_name
```

where `pool_name` is a valid resource pool within the active configuration enabled with `pooladm(1M)`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `pooladm(1M)`, `setrctl(2)`, `settaskid(2)`, `inproj(3PROJECT)`, `pool_error(3POOL)`, `pool_set_binding(3POOL)`, `passwd(4)`, `project(4)`, `attributes(5)`

NAME	significand – significand function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double significand(double <i>x</i>);</pre>				
DESCRIPTION	The <code>significand()</code> function, along with the <code>logb(3M)</code> and <code>scalb(3M)</code> functions, allows users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California. If x equals $sig * 2^n$ with $1 < sig < 2$, then <code>significand(<i>x</i>)</code> returns <i>sig</i> for exercising the fraction-part(F) test vector. <code>significand(<i>x</i>)</code> is not defined when <i>x</i> is either 0, $\pm\text{Inf}$ or NaN.				
RETURN VALUES	For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by various Standards.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>logb(3M)</code> , <code>matherr(3M)</code> , <code>scalb(3M)</code> , <code>attributes(5)</code>				

sin(3M)

NAME	sin – sine function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double sin(double <i>x</i>);</pre>				
DESCRIPTION	The <code>sin()</code> function computes the sine of its argument <i>x</i> , measured in radians.				
RETURN VALUES	Upon successful completion, <code>sin()</code> returns the sine of <i>x</i> . If <i>x</i> is NaN or $\pm\text{Inf}$, NaN is returned.				
ERRORS	No errors will occur.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>asin(3M)</code> , <code>isnan(3M)</code> , <code>attributes(5)</code>				

NAME	sinh – hyperbolic sine function				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double sinh(double x);</pre>				
DESCRIPTION	The <code>sinh()</code> function computes the hyperbolic sine of x .				
RETURN VALUES	Upon successful completion, <code>sinh()</code> returns the hyperbolic sine of x. If the result would cause an overflow, <code>±HUGE_VAL</code> is returned and <code>errno</code> is set to <code>ERANGE</code>. If x is NaN, NaN is returned. For exceptional cases, <code>matherr(3M)</code> tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The <code>sinh()</code> function will fail if: <code>ERANGE</code> The result would cause overflow.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>sinh()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>asinh(3M)</code> , <code>cosh(3M)</code> , <code>isnan(3M)</code> , <code>matherr(3M)</code> , <code>tanh(3M)</code> , <code>attributes(5)</code> , <code>standards(5)</code>				

sqrt(3M)

NAME	sqrt – square root function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double sqrt(double <i>x</i>);</pre>				
DESCRIPTION	The <code>sqrt()</code> function computes the square root of <i>x</i> .				
RETURN VALUES	Upon successful completion, <code>sqrt()</code> returns the square root of <i>x</i>. If <i>x</i> is NaN, NaN is returned. If <i>x</i> is negative, NaN is returned and <code>errno</code> is set to <code>EDOM</code>.				
ERRORS	The <code>sqrt()</code> function will fail if: <table><tr><td>EDOM</td><td>The value of <i>x</i> is negative.</td></tr></table>	EDOM	The value of <i>x</i> is negative.		
EDOM	The value of <i>x</i> is negative.				
USAGE	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>sqrt()</code> . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>isnan(3M)</code> , <code>attributes(5)</code>				

SSAAgentIsAlive(3SNMP)

NAME	SSAAgentIsAlive, SSAGetTrapPort, SSARegSubtable, SSARegSubagent, SSARegSubtree, SSASendTrap, SSASubagentOpen – Sun Solstice Enterprise Agent registration and communication helper functions
SYNOPSIS	<pre>cc [flag ...] file ... -lssagent -lssasnmplib [library ..] #include <impl.h> extern int SSAAgentIsAlive(IPAddress *agent_addr, int *port, char *community, struct timeval *timeout); extern int SSAGetTrapPort(); extern int *SSARegSubagent(Agent* agent); int SSARegSubtable(SSA_Table *table); int SSARegSubtree(SSA_Subtree *subtree); extern void SSASendTrap(char *name); extern int SSASubagentOpen(int *num_of_retry, char *agent_name);</pre>
DESCRIPTION	The SSAAgentIsAlive() function returns TRUE if the master agent is alive, otherwise returns FALSE. The <i>agent_addr</i> parameter is the address of the agent. Specify the security token in the <i>community</i> parameter. You can specify the maximum amount of time to wait for a response with the <i>timeout</i> parameter. The SSAGetTrapPort() function returns the port number used by the Master Agent to communicate with the subagent. The SSARegSubagent() function enables a subagent to register and unregister with a Master Agent. The <i>agent</i> parameter is a pointer to an Agent structure containing the following members: <pre>int timeout; /* optional */ int agent_id; /* required */ int agent_status; /* required */ char *personal_file; /* optional */ char *config_file; /* optional */ char *executable; /* optional */ char *version_string; /* optional */ char *protocol; /* optional */ int process_id; /* optional */ char *name; /* optional */ int system_up_time; /* optional */ int watch_dog_time; /* optional */ Address address; /* required */ struct _Agent; /* reserved */ struct _Subtree; /* reserved */</pre> The <i>agent_id</i> member is an integer value returned by the SSASubagentOpen() function. After calling SSASubagentOpen(), you pass the <i>agent_id</i> in the SSARegSubagent() call to register the subagent with the Master Agent. The following values are supported for <i>agent_status</i>:

SSAAgentIsAlive(3SNMP)

```
SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
SSA_OPER_STATUS_DESTROY
```

You pass `SSA_OPER_STATUS_DESTROY` as the value in a `SSARegSubagent()` function call when you want to unregister the agent from the Master Agent.

`Address` has the same structure as `sockaddr_in`, that is a common UNIX structure containing the following members:

```
short      sin_family;
ushort_t   sin_port;
struct     in_addr sin_addr;
char       sin_zero[8];
```

The `SSARegSubtable()` function registers a MIB table with the Master Agent. If this function is successful, an index number is returned, otherwise 0 is returned. The *table* parameter is a pointer to a `SSA_Table` structure containing the following members:

```
int  regTblIndex;           /* index value */
int  regTblAgentID;         /* current agent ID */
Oid  regTblOID;             /* Object ID of the table */
int  regTblStartColumn;     /* start column index */
int  regTblEndColumn;       /* end column index */
int  regTblStartRow;        /* start row index */
int  regTblEndRow;          /* end row index */
int  regTblStatus;          /* status */
```

The `regTblStatus` can have one of the following values:

```
SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
```

The `SSARegSubtree()` function registers a MIB subtree with the master agent. If successful this function returns an index number, otherwise 0 is returned. The *subtree* parameter is a pointer to a `SSA_Subtree` structure containing the following members:

```
int  regTreeIndex;         /* index value */
int  regTreeAgentID;        /* current agent ID */
Oid  name;                  /* Object ID to register */
int  regtreeStatus;        /* status */
```

The `regtreeStatus` can have one of the following values:

```
SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
```

The `SSASendTrap()` function instructs the Master Agent to send a trap notification, based on the keyword passed with *name*. When your subagent MIB is compiled by `mibcodegen`, it creates a lookup table of the trap notifications defined in the MIB. By passing the name of the trap notification type as *name*, the subagent instructs the Master Agent to construct the type of trap defined in the MIB.

SSAAgentIsAlive(3SNMP)

The `SSASubagentOpen()` function initializes communication between the subagent and the Master Agent. You must call this function before calling `SSARegSubagent()` to register the subagent with the Master Agent. The `SSASubagentOpen()` function returns a unique agent ID that is passed in the `SSARegSubagent()` call to register the subagent. If 0 is returned as the agent ID, the attempt to initialize communication with the Master Agent was unsuccessful. Since UDP is used to initialize communication with the Master Agent, you may want to set the value of *num_of_retry* to make multiple attempts.

The value for *agent_name* must be unique within the domain for which the Master Agent is responsible.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `attributes(5)`

SSAOidCmp(3SNMP)

NAME	SSAOidCmp, SSAOidCpy, SSAOidDup, SSAOidFree, SSAOidInit, SSAOidNew, SSAOidString, SSAOidStrToOid, SSAOidZero – Sun Solstice Enterprise Agent OID helper functions						
SYNOPSIS	<pre>cc [flag ...] file ... -lssasnmplib [library ..] #include <impl.h> int SSAOidCmp(Oid *oid1, Oid *oid2); int SSAOidCpy(Oid *oid1, Oid *oid2, char *error_label); Oid *SSAOidDup(Oid *oid, char *error_label); void SSAOidFree(Oid *oid); int SSAOidInit(Oid *oid, Subid *subids, int len, char *error_label); Oid *SSAOidNew(); char *SSAOidString(Oid *oid); Oid *SSAOidStrToOid(char* name, char *error_label); void SSAOidZero(Oid *oid);</pre>						
DESCRIPTION	The SSAOidCmp() function performs a comparison of the given OIDs. This function returns: <table> <tr> <td>0</td> <td>if <i>oid1</i> is equal to <i>oid2</i></td> </tr> <tr> <td>1</td> <td>if <i>oid1</i> is greater than <i>oid2</i></td> </tr> <tr> <td>-1</td> <td>if <i>oid1</i> is less than <i>oid2</i></td> </tr> </table> The SSAOidCpy() function makes a deep copy of <i>oid2</i> to <i>oid1</i>. This function assumes <i>oid1</i> has been processed by the SSAOidZero() function. Memory is allocated inside <i>oid1</i> and the contents of <i>oid2</i>, not just the pointer, is copied to <i>oid1</i>. If an error is encountered, an error message is stored in the <i>error_label</i> buffer. The SSAOidDup() function returns a clone of <i>oid</i>, by using the deep copy. Error information is stored in the <i>error_label</i> buffer. The SSAOidFree() function frees the OID instance, with its content. The SSAOidNew() function returns a new OID. The SSAOidInit() function copies the Subid array from <i>subids</i> to the OID instance with the specified length <i>len</i>. This function assumes that the OID instance has been processed by the SSAOidZero() function or no memory is allocated inside the OID instance. If an error is encountered, an error message is stored in the <i>error_label</i> buffer. The SSAOidString() function returns a char pointer for the printable form of the given <i>oid</i>.	0	if <i>oid1</i> is equal to <i>oid2</i>	1	if <i>oid1</i> is greater than <i>oid2</i>	-1	if <i>oid1</i> is less than <i>oid2</i>
0	if <i>oid1</i> is equal to <i>oid2</i>						
1	if <i>oid1</i> is greater than <i>oid2</i>						
-1	if <i>oid1</i> is less than <i>oid2</i>						

SSAOidCmp(3SNMP)

The `SSAOidStrToOid()` function returns a new OID instance from *name*. If an error is encountered, an error message is stored in the *error_label* buffer.

The `SSAOidZero()` function frees the memory used by the OID object for buffers, but not the OID instance itself.

RETURN VALUES The `SSAOidNew()` and `SSAOidStrToOid()` functions return 0 if an error is detected.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `attributes(5)`

SSAStringCpy(3SNMP)

NAME	SSAStringCpy, SSAStringInit, SSAStringToChar, SSAStringZero – Sun Solstice Enterprise Agent string helper functions				
SYNOPSIS	<pre>cc [flag ...] file ... -lssasnmp [library ..] #include <impl.h> void *SSAStringZero(String *string); int SSAStringInit(String *string, uchar_t *chars, int len, char *error_label); int SSAStringCpy(String *string1, String *string2, char *error_label); char *SSAStringToChar(String string);</pre>				
DESCRIPTION	The SSAStringCpy() function makes a deep copy of <i>string2</i> to <i>string1</i>. This function assumes that <i>string1</i> has been processed by the SSAStringZero() function. Memory is allocated inside the <i>string1</i> and the contents of <i>string2</i>, not just the pointer, is copied to the <i>string1</i>. If an error is encountered, an error message is stored in the <i>error_label</i> buffer. The SSAStringInit() function copies the char array from <i>chars</i> to the string instance with the specified length <i>len</i>. This function assumes that the string instance has been processed by the SSAStringZero() function or no memory is allocated inside the string instance. If an error is encountered, an error message is stored in the <i>error_label</i> buffer. The SSAStringToChar() function returns a temporary char array buffer for printing purposes. The SSAStringZero() function frees the memory inside of the String instance, but not the string object itself.				
RETURN VALUES	The SSAStringInit() and SSAStringCpy() functions return 0 if successful and -1 if error.				
ATTRIBUTES	See attributes (5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	attributes(5)				

NAME	strccpy, streadd, strcadd, strecpy – copy strings, compressing or expanding escape codes				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *strccpy(char *<i>output</i>, const char *<i>input</i>) ; char *strcadd(char *<i>output</i>, const char *<i>input</i>) ; char *strecpy(char *<i>output</i>, const char *<i>input</i>, const char *<i>exceptions</i>) ; char *streadd(char *<i>output</i>, const char *<i>input</i>, const char *<i>exceptions</i>) ;</pre>				
DESCRIPTION	strccpy() copies the <i>input</i> string, up to a null byte, to the <i>output</i> string, compressing the C-language escape sequences (for example, <code>\n</code>, <code>\001</code>) to the equivalent character. A null byte is appended to the output. The <i>output</i> argument must point to a space big enough to accommodate the result. If it is as big as the space pointed to by <i>input</i> it is guaranteed to be big enough. strccpy() returns the <i>output</i> argument. strcadd() is identical to strccpy(), except that it returns the pointer to the null byte that terminates the output. strecpy() copies the <i>input</i> string, up to a null byte, to the <i>output</i> string, expanding non-graphic characters to their equivalent C-language escape sequences (for example, <code>\n</code>, <code>\001</code>). The <i>output</i> argument must point to a space big enough to accommodate the result; four times the space pointed to by <i>input</i> is guaranteed to be big enough (each character could become <code>\</code> and 3 digits). Characters in the <i>exceptions</i> string are not expanded. The <i>exceptions</i> argument may be zero, meaning all non-graphic characters are expanded. strecpy() returns the <i>output</i> argument. streadd() is identical to strecpy(), except that it returns the pointer to the null byte that terminates the output.				
EXAMPLES	EXAMPLE 1 Example of expanding and compressing escape codes. <pre>/* expand all but newline and tab */ strecpy(output, input, "\n\t"); /* concatenate and compress several strings */ cp = strcadd(output, input1); cp = strcadd(cp, input2); cp = strcadd(cp, input3);</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	string(3C) , strfind(3GEN) , attributes(5)				

strccpy(3GEN)

NOTES	When compiling multi-thread applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multi-thread applications.

NAME	strfind, strstrpn, strtrns, str – string manipulations				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> int strfind(const char *<i>as1</i>, const char *<i>as2</i>); char *strstrpn(const char *<i>string</i>, const char *<i>tc</i>); char * strtrns(const char *<i>string</i>, const char *<i>old</i>, const char *<i>new</i>, char *<i>result</i>);</pre>				
DESCRIPTION	The <code>strfind()</code> function returns the offset of the first occurrence of the second string, <i>as2</i>, if it is a substring of string <i>as1</i>. If the second string is not a substring of the first string <code>strfind()</code> returns -1. The <code>strstrpn()</code> function trims chartacters from a string. It searches from the end of <i>string</i> for the first character that is not contained in <i>tc</i>. If such a character is found, <code>strstrpn()</code> returns a pointer to the next character; otherwise, it returns a pointer to <i>string</i>. The <code>strtrns()</code> function transforms <i>string</i> and copies it into <i>result</i>. Any character that appears in <i>old</i> is replaced with the character in the same position in <i>new</i>. The <i>new</i> result is returned.				
USAGE	When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multithreaded applications.				
EXAMPLES	EXAMPLE 1 An example of the <code>strfind()</code> function. <pre>/* find offset to substring "hello" within as1 */ i = strfind(as1, "hello"); /* trim junk from end of string */ s2 = strstrpn(s1, ".*?#\$\$%"); *s2 = '\0'; /* transform lower case to upper case */ a1[] = "abcdefghijklmnopqrstuvwxyz"; a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; s2 = strtrns(s1, a1, a2, s2);</pre>				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>string(3C)</code> , <code>attributes(5)</code>				

sysevent_bind_handle(3SYSEVENT)

NAME	sysevent_bind_handle, sysevent_unbind_handle – bind or unbind subscriber handle										
SYNOPSIS	<pre>cc [flag...] file... -lsysevent [library...] #include <libsysevent.h> sysevent_handle_t *sysevent_bind_handle(void (*event_handler) (sysevent_t *ev), void *cookie); void sysevent_unbind_handle(sysevent_handle_t *sysevent_hdl);</pre>										
PARAMETERS	<i>ev</i> pointer to sysevent buffer handle <i>event_handler</i> pointer to an event handling function <i>sysevent_hdl</i> pointer to a sysevent subscriber handle										
DESCRIPTION	The <code>sysevent_bind_handle()</code> function allocates memory associated with a subscription handle and binds it to the caller's <i>event_handler</i>. The <i>event_handler</i> is invoked during subsequent system event notifications once a subscription has been made with <code>sysevent_subscribe_event(3SYSEVENT)</code>. The system event is represented by the argument <i>ev</i> and is passed as an argument to the invoked event delivery function, <i>event_handler</i>. Additional threads are created to service communication between <code>syseventd(1M)</code> and the calling process and to run the event handler routine, <i>event_handler</i>. The <code>sysevent_unbind_handle()</code> function deallocates memory and other resources associated with a subscription handle and deactivates all system event notifications for the calling process. All event notifications are guaranteed to stop upon return from <code>sysevent_unbind_handle()</code>.										
RETURN VALUES	The <code>sysevent_bind_handle()</code> function returns a valid sysevent subscriber handle if the handle is successfully allocated. Otherwise, NULL is returned and <code>errno</code> is set to indicate the error. The <code>sysevent_unbind_handle()</code> function returns no value.										
ERRORS	The <code>sysevent_bind_handle()</code> function will fail if: <table> <tr> <td>EACCESS</td><td>The calling process has an ID other than the privileged user.</td></tr> <tr> <td>EBUSY</td><td>There are no resources available.</td></tr> <tr> <td>EINVAL</td><td>The pointer to the function <i>event_handler</i> is NULL.</td></tr> <tr> <td>EMFILE</td><td>The process has too many open descriptors.</td></tr> <tr> <td>ENOMEM</td><td>There are insufficient resources to allocate the handle.</td></tr> </table>	EACCESS	The calling process has an ID other than the privileged user.	EBUSY	There are no resources available.	EINVAL	The pointer to the function <i>event_handler</i> is NULL.	EMFILE	The process has too many open descriptors.	ENOMEM	There are insufficient resources to allocate the handle.
EACCESS	The calling process has an ID other than the privileged user.										
EBUSY	There are no resources available.										
EINVAL	The pointer to the function <i>event_handler</i> is NULL.										
EMFILE	The process has too many open descriptors.										
ENOMEM	There are insufficient resources to allocate the handle.										

sysevent_bind_handle(3SYSEVENT)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO syseventd(1M), sysevent_subscribe_event(3SYSEVENT), attributes(5)

NOTES Event notifications are revoked by syseventd when the bound process dies. Event notification is suspended if a signal is caught and handled by the event_handler thread. Event notification is also suspended when the calling process attempts to use fork(2) or fork1(2). Event notifications might be lost during suspension periods.

sysevent_free(3SYSEVENT)

NAME	sysevent_free – free memory for sysevent handle
SYNOPSIS	<pre>cc [flag...] file...-lsysevent [library...] #include <libsysevent.h> void sysevent_free(sysevent_t *ev);</pre>
PARAMETERS	<i>ev</i> handle to event an event buffer
DESCRIPTION	The <code>sysevent_free()</code> function deallocates memory associated with an event buffer.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO `attributes(5)`

sysevent_get_attr_list(3SYSEVENT)

NAME	sysevent_get_attr_list – get attribute list pointer						
SYNOPSIS	<pre>cc [flag...] file... -lsysevent -lnvpair [library...] #include <libsysevent.h> #include <libnvpair.h> int sysevent_get_attr_list(sysevent_t *ev, nvlist_t **attr_list);</pre>						
PARAMETERS	<i>ev</i> handle to a system event <i>attr_list</i> address of a pointer to attribute list (nvlist_t)						
DESCRIPTION	The <code>sysevent_get_attr_list()</code> function updates <i>attr_list</i> to point to a searchable name-value pair list associated with the <code>sysevent</code> event, <i>ev</i> . The interface manages the allocation of the attribute list, but it is up to the caller to free the list when it is no longer needed with a call to <code>nvlist_free()</code> . See <code>nvlist_alloc(3NVPAR)</code> .						
RETURN VALUES	The <code>sysevent_get_attr_list()</code> function returns 0 if the attribute list for <i>ev</i> is found to be valid. Otherwise it returns -1 and sets <code>errno</code> to indicate the error.						
ERRORS	The <code>sysevent_get_attr_list()</code> function will fail if: ENOMEM Insufficient memory available to allocate an nvlist. EINVAL Invalid sysevent event attribute list.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Interface Stability</td><td>Evolving</td></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Interface Stability	Evolving	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Interface Stability	Evolving						
MT-Level	MT-Safe						
SEE ALSO	<code>syseventd(1M)</code> , <code>nvlist_alloc(3NVPAR)</code> , <code>nvlist_lookup_boolean(3NVPAR)</code> , <code>attributes(5)</code>						

sysevent_get_class_name(3SYSEVENT)

NAME	sysevent_get_class_name, sysevent_get_subclass_name, sysevent_get_event_id, sysevent_get_size – get class name, subclass name, ID or buffer size of event
SYNOPSIS	<pre>cc [flag...] file...-lsysevent [library...] #include <libsysevent.h> char *sysevent_get_class_name(sysevent_t *ev); char *sysevent_get_subclass_name(sysevent_t *ev); void sysevent_get_event_id(sysevent_t *ev, sysevent_id_t *eid); int sysevent_get_size(sysevent_t *ev);</pre>
PARAMETERS	<i>eid</i> pointer to sysevent_id_t structure <i>ev</i> handle to event
DESCRIPTION	The sysevent_get_class_name() and sysevent_get_subclass_name() functions return, respectively, the class and subclass names for the provided event <i>ev</i>. The sysevent_get_event_id() function returns the unique event identifier associated with the sysevent handle, <i>ev</i>. The identifier is composed of a relative timestamp issued at the time the event was generated and a sequence number to ensure uniqueness. <pre>typedef struct sysevent_id { uint64_t eid_seq; hrtime_t eid_ts; } sysevent_id_t;</pre> The sysevent_get_size() function returns the size of the event buffer, <i>ev</i>.
EXAMPLES	EXAMPLE 1 Parse sysevent header information. The following example parses sysevent header information from an application's event handler. <pre>hrtime_t last_ev_time; unit64_t last_ev_seq; void event_handler(sysevent_t *ev) { sysevent_t *new_ev; sysevent_id_t eid; /* Filter on class and subclass */ if (strcmp(EC_PRIV, sysevent_get_class_name(ev)) != 0) { return; } else if (strcmp("ESC_MYSUBCLASS", sysevent_get_subclass_name(ev)) != 0) { return; } }</pre>

EXAMPLE 1 Parse sysevent header information. *(Continued)*

```
/*
 * Check for replayed sysevent, time must
 * be greater than previously recorded.
 */
sysevent_get_event_id(ev, &eid);
if (eid.eid_ts < last_ev_time ||
    (eid.eid_ts == last_ev_time && eid.eid_seq <=
     last_ev_seq)) {
    return;
}

last_ev_time = eid.eid_ts;
last_ev_seq = eid.eid_seq;

/* Store event for later processing */
ev_sz = sysevent_get_size(ev);
new_ev (sysevent_t *)malloc(ev_sz);
bcopy(ev, new_ev, ev_sz);
queue_event(new_ev);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO attributes(5)

sysevent_get_vendor_name(3SYSEVENT)

NAME	sysevent_get_vendor_name, sysevent_get_pub_name, sysevent_get_pid – get vendor name, publisher name or processor ID of event
SYNOPSIS	<pre>cc [flag ...] file ...-lsysevent [library ...] #include <libsysevent.h> char *sysevent_get_vendor_name(sysevent_t *ev); char *sysevent_get_pub_name(sysevent_t *ev); pid_t sysevent_get_pid(sysevent_t *ev);</pre>
PARAMETERS	<i>ev</i> handle to a system event object
DESCRIPTION	The sysevent_get_pub_name() function returns the publisher name for the sysevent handle, <i>ev</i>. The publisher name identifies the name of the publishing application or kernel subsystem of the sysevent. The sysevent_get_pid() function returns the process ID for the publishing application or SE_KERN_PID for sysevents originating in the kernel. The publisher name and PID are useful for implementing event acknowledgement. The sysevent_get_vendor_name() function returns the vendor string for the publishing application or kernel subsystem. A vendor string is the company's stock symbol that provided the application or kernel subsystem that generated the system event. This information is useful for filtering sysevents for one or more vendors. The interface manages the allocation of the vendor and publisher name strings, but it is up to the caller to free the strings when they are no longer needed with a call to free(). See malloc(3MALLOC).
EXAMPLES	EXAMPLE 1 Parse sysevent header information. The following example parses sysevent header information from an application's event handler. <pre>char *vendor; char *pub; void event_handler(sysevent_t *ev) { if (strcmp(EC_PRIV, sysevent_get_class_name(ev)) != 0) { return; } vendor = sysevent_get_vendor_name(ev); if (strcmp("SUNW", vendor) != 0) { free(vendor); return; } pub = sysevent_get_pub_name(ev); if (strcmp("test_daemon", pub) != 0) { free(vendor); free(pub); } }</pre>

sysevent_get_vendor_name(3SYSEVENT)

EXAMPLE 1 Parse sysevent header information. (Continued)

```
        return;
    }
    (void) kill(sysevent_get_pid(ev), SIGUSR1);
    free(vendor);
    free(pub);
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO malloc(3MALLOC), attributes(5)

sysevent_post_event(3SYSEVENT)

NAME	sysevent_post_event – post system event for applications												
SYNOPSIS	<pre>cc [flag...] file ...-lsysevent -lnvpair [library...] #include <libsysevent.h> #include <libnvpair.h> int sysevent_post_event(char *class, char *subclass, char *vendor, char *publisher, nvlist_t *attr_list, sysevent_id_t *eid);</pre>												
PARAMETERS	<table> <tr> <td><i>attr_list</i></td><td>pointer to an nvlist_t, listing the name-value attributes associated with the event, or NULL if there are no such attributes for this event</td></tr> <tr> <td><i>class</i></td><td>pointer to a string defining the event class</td></tr> <tr> <td><i>eid</i></td><td>pointer to a system unique identifier</td></tr> <tr> <td><i>publisher</i></td><td>pointer to a string defining the event's publisher name</td></tr> <tr> <td><i>subclass</i></td><td>pointer to a string defining the event subclass</td></tr> <tr> <td><i>vendor</i></td><td>pointer to a string defining the vendor</td></tr> </table>	<i>attr_list</i>	pointer to an nvlist_t, listing the name-value attributes associated with the event, or NULL if there are no such attributes for this event	<i>class</i>	pointer to a string defining the event class	<i>eid</i>	pointer to a system unique identifier	<i>publisher</i>	pointer to a string defining the event's publisher name	<i>subclass</i>	pointer to a string defining the event subclass	<i>vendor</i>	pointer to a string defining the vendor
<i>attr_list</i>	pointer to an nvlist_t, listing the name-value attributes associated with the event, or NULL if there are no such attributes for this event												
<i>class</i>	pointer to a string defining the event class												
<i>eid</i>	pointer to a system unique identifier												
<i>publisher</i>	pointer to a string defining the event's publisher name												
<i>subclass</i>	pointer to a string defining the event subclass												
<i>vendor</i>	pointer to a string defining the vendor												
DESCRIPTION	The <code>sysevent_post_event()</code> function causes a system event of the specified class, subclass, vendor, and publisher to be generated on behalf of the caller and queued for delivery to the sysevent daemon <code>syseventd(1M)</code>. The vendor must be the company stock symbol of the event posting application. The publisher should be the name of the application generating the event. For example, all events posted by Sun applications begin with the company's stock symbol, "SUNW". The publisher is usually the name of the application generating the system event. A system event generated by <code>devfsadm(1M)</code> has a publisher string of <code>devfsadm</code>. The publisher information is used by sysevent consumers to filter unwanted event publishers. Upon successful queuing of the system event, a unique identifier is assigned to <i>eid</i>.												
RETURN VALUES	The <code>sysevent_post_event()</code> function returns 0 if the system event has been queued successfully for delivery. Otherwise it returns -1 and sets <code>errno</code> to indicate the error.												
ERRORS	The <code>sysevent_post_event()</code> function will fail if: <table> <tr> <td>ENOMEM</td><td>Insufficient resources to queue the system event.</td></tr> <tr> <td>EIO</td><td>The <code>syseventd</code> daemon is not responding and events cannot be queued or delivered at this time.</td></tr> <tr> <td>EINVAL</td><td>Invalid argument.</td></tr> <tr> <td>EPERM</td><td>Permission denied.</td></tr> </table>	ENOMEM	Insufficient resources to queue the system event.	EIO	The <code>syseventd</code> daemon is not responding and events cannot be queued or delivered at this time.	EINVAL	Invalid argument.	EPERM	Permission denied.				
ENOMEM	Insufficient resources to queue the system event.												
EIO	The <code>syseventd</code> daemon is not responding and events cannot be queued or delivered at this time.												
EINVAL	Invalid argument.												
EPERM	Permission denied.												

sysevent_post_event(3SYSEVENT)

EFAULT A copy error occurred.

EXAMPLES

EXAMPLE 1 Post a system event event with no attributes.

The following example posts a system event event with no attributes.

```
if (sysevent_post_event(EC_PRIV, "ESC_MYSUBCLASS", "SUNW", argv[0],
    NULL) != 0) {
    fprintf(stdout, "error logging system event\n");
}
```

EXAMPLE 2 Post a system event with two name-value pair attributes.

The following example posts a system event event with two name-value pair attributes, an integer value and a string.

```
nvlist_t      *attr_list;
uint32_t      uint32_val = 0xFFFFFFFF;
char          *string_val = "string value data";

if (nvlist_alloc(&attr_list, 0, 0) == 0) {
    err = nvlist_add_uint32(attr_list, "uint32 data", uint32_val);
    if (err == 0)
        err = nvlist_add_string(attr_list, "str data",
            string_val);
    if (err == 0)
        err = sysevent_post_event("EC_PRIV", "ESC_MYSUBCLASS",
            "SUNW", argv[0], attr_list);
    if (err != 0)
        fprintf(stdout, "error logging system event\n");
    nvlist_free(attr_list);
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

devfsadm(1M), syseventd(1M), nvlist_add_boolean(3NVPAIR),
nvlist_alloc(3NVPAIR), attributes(5)

sysevent_subscribe_event(3SYSEVENT)

NAME	sysevent_subscribe_event, sysevent_unsubscribe_event – register or unregister interest in event receipt
SYNOPSIS	<pre>cc [flag ...] file ... -l [library ...] #include <libsysevent.h> int sysevent_subscribe_event(sysevent_handle_t *sysevent_hdl, char *event_class, char **event_subclass_list, int num_subclasses); void sysevent_unsubscribe_event(sysevent_handle_t *sysevent_hdl, char *event_class);</pre>
PARAMETERS	<i>event_class</i> system event class string <i>event_subclass_list</i> array of subclass strings <i>num_subclasses</i> number of subclass strings <i>sysevent_hdl</i> sysevent subscriber handle
DESCRIPTION	The <code>sysevent_subscribe_event()</code> function registers the caller's interest in event notifications belonging to the class <i>event_class</i> and the subclasses contained in <i>event_subclass_list</i>. The subscriber handle <i>sysevent_hdl</i> is updated with the new subscription and the calling process receives event notifications from the event handler specified in <i>sysevent_bind_handle</i>. System events matching <i>event_class</i> and a subclass contained in <i>event_subclass_list</i> published after the caller returns from <code>sysevent_subscribe_event()</code> are guaranteed to be delivered to the calling process. Matching system events published and queued prior to a call to <code>sysevent_subscribe_event()</code> may be delivered to the process's event handler. The <i>num_subclasses</i> argument provides the number of subclass string elements in <i>event_subclass_list</i>. A caller can use the event class <code>SE_ALL_CLASSES</code> to subscribe to all event classes and subclasses. The event class <code>SE_ALL_SUBCLASSES</code> can be used to subscribe to all subclasses within a given event class. Subsequent calls to <code>sysevent_subscribe_event()</code> are allowed to add additional classes or subclasses. To remove an existing subscription, <code>sysevent_unsubscribe_event()</code> must be used to remove the subscription. The <code>sysevent_unsubscribe_event()</code> function removes the subscription described by <i>event_class</i> for <i>sysevent_hdl</i>. Event notifications matching <i>event_class</i> will not be delivered to the calling process upon return. A caller can use the event class <code>SE_ALL_CLASSES</code> to remove all subscriptions for <i>sysevent_hdl</i>. The library manages all subscription resources.

sysevent_subscribe_event(3SYSEVENT)

RETURN VALUES

The `sysevent_subscribe_event()` function returns 0 if the subscription is successful. Otherwise, -1 is returned and `errno` is set to indicate the error.

The `sysevent_unsubscribe_event()` function returns no value.

ERRORS

The `sysevent_subscribe_event()` function will fail if:

EACCESS	The calling process has an ID other than the privileged user.
EINVAL	The <i>sysevent_hdl</i> argument is an invalid <i>sysevent</i> handle.
ENOMEM	There is insufficient memory available to allocate subscription resources.

EXAMPLES

EXAMPLE 1 Subscribing for DR and environmental events

```
/* ARGUSED */
void
event_handler(sysevent_t *ev)
{
    int32_t state;
    nvlist_t *nv_list;

    if (strcmp(EC_DR, sysevent_get_class_name(ev)) == 0 &&
        strcmp(ESC_AP_STATE_CHANGE,
            sysevent_get_subclass_name(ev)) == 0) {

        /* Initiate DR activity */

        return;
    }

    if (strcmp(EC_ENV, sysevent_get_class_name(ev)) == 0) {
        if (sysevent_get_attr_list(ev, &nvlist) != 0) {
            return;
        }

        if (nvlist_lookup_int32(nvlist, ENV_FRU_STATE,
            &state) != 0) {
            nvlist_free(nvlist);
            return;
        }

        if (state == ENV_FAILED) {

            /* Initiate FRU shutdown */

        }
        nvlist_free(nvlist);
    }
}

#define MAX_SUBCLASS 3

void
main(int argc, char **argv)
```

sysevent_subscribe_event(3SYSEVENT)

EXAMPLE 1 Subscribing for DR and environmental events (Continued)

```
{
    sysevent_handle_t *shp;
    char *subclass_list[MAX_SUBCLASS];

    /* Bind event handler and create subscriber handle */
    shp = sysevent_bind_handle(event_handler);
    if (shp == NULL)
        exit(1);

    /* Subscribe to all DR event notifications */
    subclass_list[0] = SE_ALL_SUBCLASSES;
    if (sysevent_subscribe_event(shp, EC_DR, &subclass_list,
        1) != 0) {
        sysevent_unbind_handle(shp);
        exit(1);
    }

    /* Subscribe to power, fan, and temp environmental alerts */
    subclass_list[0] = ESC_ENV_POWER;
    subclass_list[1] = ESC_ENV_FAN;
    subclass_list[2] = ESC_ENV_TEMP;
    if (sysevent_subscribe_event(shp, EC_ENV, &subclass_list,
        MAX_SUBCLASS) != 0) {
        sysevent_unbind_handle(shp);
        exit(1);
    }

    for (;;) {
        (void) pause();
    }
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving
MT-Level	MT-Safe

SEE ALSO

syseventd(1M), sysevent_bind_handle(3SYSEVENT),
sysevent_get_attr_list(3SYSEVENT),
sysevent_get_class_name(3SYSEVENT),
sysevent_get_vendor_name(3SYSEVENT), attributes(5)

NAME	tan – tangent function
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double tan(double <i>x</i>);</pre>
DESCRIPTION	The <code>tan()</code> function computes the tangent of its argument <i>x</i> , measured in radians.
RETURN VALUES	Upon successful completion, <code>tan()</code> returns the tangent of <i>x</i> . If <i>x</i> is NaN or $\pm\text{Inf}$, NaN is returned.
ERRORS	No errors will occur.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO `atan(3M)`, `isnan(3M)`, `attributes(5)`

tanh(3M)

NAME	tanh – hyperbolic tangent function				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lm [<i>library</i> ...] #include <math.h> double tanh(double <i>x</i>);</pre>				
DESCRIPTION	The <code>tanh()</code> function computes the hyperbolic tangent of <i>x</i> .				
RETURN VALUES	Upon successful completion, <code>tanh()</code> returns the hyperbolic tangent of <i>x</i> . If <i>x</i> is NaN, NaN is returned.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	<code>atanh(3M)</code> , <code>isnan(3M)</code> , <code>tan(3M)</code> , <code>attributes(5)</code>				

tnfctl_buffer_alloc(3TNF)

NAME	tnfctl_buffer_alloc, tnfctl_buffer_dealloc – allocate or deallocate a buffer for trace data						
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_buffer_alloc(tnfctl_handle_t *hndl, const char *trace_file_name, size_t trace_buffer_size); tnfctl_buffer_dealloc(tnfctl_handle_t *hndl);</pre>						
DESCRIPTION	tnfctl_buffer_alloc() allocates a buffer to which trace events are logged. When tracing a process using a tnfctl handle returned by tnfctl_pid_open(3TNF), tnfctl_exec_open(3TNF), tnfctl_indirect_open(3TNF), and tnfctl_internal_open(3TNF), <i>trace_file_name</i> is the name of the trace file to which trace events should be logged. It can be an absolute path specification or a relative path specification. If it is relative, the current working directory of the process that is calling tnfctl_buffer_alloc() is prefixed to <i>trace_file_name</i>. If the named trace file already exists, it is overwritten. For kernel tracing, that is, for a tnfctl handle returned by tnfctl_kernel_open(3TNF), trace events are logged to a trace buffer in memory; therefore, <i>trace_file_name</i> is ignored. Use tnfextract(1) to extract a kernel buffer into a file. <i>trace_buffer_size</i> is the size in bytes of the trace buffer that should be allocated. An error is returned if an attempt is made to allocate a buffer when one already exists. tnfctl_buffer_alloc() affects the trace attributes; use tnfctl_trace_attrs_get(3TNF) to get the latest trace attributes after a buffer is allocated. tnfctl_buffer_dealloc() is used to deallocate a kernel trace buffer that is no longer needed. <i>hndl</i> must be a kernel handle, returned by tnfctl_kernel_open(3TNF). A process's trace file cannot be deallocated using tnfctl_buffer_dealloc(). Instead, once the trace file is no longer needed for analysis and after the process being traced exits, use rm(1) to remove the trace file. Do not remove the trace file while the process being traced is still alive. tnfctl_buffer_dealloc() affects the trace attributes; use tnfctl_trace_attrs_get(3TNF) to get the latest trace attributes after a buffer is deallocated. For a complete discussion of tnf tracing, see tracing(3TNF).						
RETURN VALUES	tnfctl_buffer_alloc() and tnfctl_buffer_dealloc() return TNFCTL_ERR_NONE upon success.						
ERRORS	The following error codes apply to tnfctl_buffer_alloc(): <table><tr><td>TNFCTL_ERR_BUFEXISTS</td><td>A buffer already exists.</td></tr><tr><td>TNFCTL_ERR_ACCES</td><td>Permission denied; could not create a trace file.</td></tr><tr><td>TNFCTL_ERR_SIZETOOSMALL</td><td>The <i>trace_buffer_size</i> requested is smaller than the minimum trace buffer size needed.</td></tr></table>	TNFCTL_ERR_BUFEXISTS	A buffer already exists.	TNFCTL_ERR_ACCES	Permission denied; could not create a trace file.	TNFCTL_ERR_SIZETOOSMALL	The <i>trace_buffer_size</i> requested is smaller than the minimum trace buffer size needed.
TNFCTL_ERR_BUFEXISTS	A buffer already exists.						
TNFCTL_ERR_ACCES	Permission denied; could not create a trace file.						
TNFCTL_ERR_SIZETOOSMALL	The <i>trace_buffer_size</i> requested is smaller than the minimum trace buffer size needed.						

tnfctl_buffer_alloc(3TNF)

Use `trace_min_size` of trace attributes in `tnfctl_trace_attrs_get(3TNF)` to determine the minimum size of the buffer.

TNFCTL_ERR_SIZETOOBIG

The requested trace file size is too big.

TNFCTL_ERR_BADARG

trace_file_name is NULL or the absolute path name is longer than MAXPATHLEN.

TNFCTL_ERR_ALLOCFAIL

A memory allocation failure occurred.

TNFCTL_ERR_INTERNAL

An internal error occurred.

The following error codes apply to `tnfctl_buffer_dealloc()`:

TNFCTL_ERR_BADARG

hndl is not a kernel handle.

TNFCTL_ERR_NOBUF

No buffer exists to deallocate.

TNFCTL_ERR_BADDEALLOC

Cannot deallocate a trace buffer unless tracing is stopped. Use `tnfctl_trace_state_set(3TNF)` to stop tracing.

TNFCTL_ERR_INTERNAL

An internal error occurred.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO

`prex(1)`, `rm(1)`, `tnfxtract(1)`, `TNF_PROBE(3TNF)`, `libtnfctl(3TNF)`, `tnfctl_exec_open(3TNF)`, `tnfctl_indirect_open(3TNF)`, `tnfctl_internal_open(3TNF)`, `tnfctl_kernel_open(3TNF)`, `tnfctl_pid_open(3TNF)`, `tnfctl_trace_attrs_get(3TNF)`, `tracing(3TNF)`, `attributes(5)`

NAME	tnfctl_close – close a tnfctl handle								
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_close(tnfctl_handle_t *hndl, tnfctl_targ_op_t action);</pre>								
DESCRIPTION	tnfctl_close() is used to close a tnfctl handle and to free up the memory associated with the handle. When the handle is closed, the tracing state and the states of the probes are not changed. tnfctl_close() can be used to close handles in any mode, that is, whether they were created by tnfctl_internal_open(3TNF), tnfctl_pid_open(3TNF), tnfctl_exec_open(3TNF), tnfctl_indirect_open(3TNF), or tnfctl_kernel_open(3TNF). The <i>action</i> argument is only used in direct mode, that is, if <i>hndl</i> was created by tnfctl_exec_open(3TNF) or tnfctl_pid_open(3TNF). In direct mode, <i>action</i> specifies whether the process will proceed, be killed, or remain suspended. <i>action</i> may have the following values: <table> <tr> <td>TNFCTL_TARG_DEFAULT</td><td>Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3TNF), but lets it continue if it was created with tnfctl_pid_open(3TNF).</td></tr> <tr> <td>TNFCTL_TARG_KILL</td><td>Kills the target process.</td></tr> <tr> <td>TNFCTL_TARG_RESUME</td><td>Allows the target process to continue.</td></tr> <tr> <td>TNFCTL_TARG_SUSPEND</td><td>Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3TNF) interface. The target process can also be continued with prun(1).</td></tr> </table>	TNFCTL_TARG_DEFAULT	Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3TNF), but lets it continue if it was created with tnfctl_pid_open(3TNF).	TNFCTL_TARG_KILL	Kills the target process.	TNFCTL_TARG_RESUME	Allows the target process to continue.	TNFCTL_TARG_SUSPEND	Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3TNF) interface. The target process can also be continued with prun(1).
TNFCTL_TARG_DEFAULT	Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3TNF), but lets it continue if it was created with tnfctl_pid_open(3TNF).								
TNFCTL_TARG_KILL	Kills the target process.								
TNFCTL_TARG_RESUME	Allows the target process to continue.								
TNFCTL_TARG_SUSPEND	Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3TNF) interface. The target process can also be continued with prun(1).								
RETURN VALUES	tnfctl_close() returns TNFCTL_ERR_NONE upon success.								
ERRORS	The following error codes apply to tnfctl_close(): <table> <tr> <td>TNFCTL_ERR_BADARG</td><td>A bad argument was sent in <i>action</i>.</td></tr> <tr> <td>TNFCTL_ERR_INTERNAL</td><td>An internal error occurred.</td></tr> </table>	TNFCTL_ERR_BADARG	A bad argument was sent in <i>action</i> .	TNFCTL_ERR_INTERNAL	An internal error occurred.				
TNFCTL_ERR_BADARG	A bad argument was sent in <i>action</i> .								
TNFCTL_ERR_INTERNAL	An internal error occurred.								
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtnfc</td></tr> <tr> <td>MT Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfc	MT Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtnfc								
MT Level	MT-Safe								

tnfctl_close(3TNF)

SEE ALSO | prex(1), prun(1), TNF_PROBE(3TNF), libtnfctl(3TNF),
tnfctl_exec_open(3TNF), tnfctl_indirect_open(3TNF),
tnfctl_kernel_open(3TNF), tnfctl_pid_open(3TNF), tracing(3TNF),
attributes(5)

tnfctl_indirect_open(3TNF)

NAME	tnfctl_indirect_open, tnfctl_check_libs – control probes of another process where caller provides /proc functionality
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_indirect_open(void *prochandle, tnfctl_ind_config_t *config, tnfctl_handle_t **ret_val); tnfctl_errcode_t tnfctl_check_libs(tnfctl_handle_t *hndl);</pre>
DESCRIPTION	The interfaces <code>tnfctl_indirect_open()</code> and <code>tnfctl_check_libs()</code> are used to control probes in another process where the <code>libtnfctl(3TNF)</code> client has already opened <code>proc(4)</code> on the target process. An example of this is when the client is a debugger. Since these clients already use <code>/proc</code> on the target, <code>libtnfctl(3TNF)</code> cannot use <code>/proc</code> directly. Therefore, these clients must provide callback functions that can be used to inspect and to update the target process. The target process must load <code>libtnfprobe.so.1</code> (defined in <code><tnf/tnfctl.h></code> as macro <code>TNFCTL_LIBTNFPROBE</code>). The first argument <i>prochandle</i> is a pointer to an opaque structure that is used in the callback functions that inspect and update the target process. This structure should encapsulate the state that the caller needs to use <code>/proc</code> on the target process (the <code>/proc</code> file descriptor). The second argument, <i>config</i>, is a pointer to <pre>typedef struct tnfctl_ind_config { int (*p_read)(void *prochandle, paddr_t addr, char *buf, size_t size); int (*p_write)(void *prochandle, paddr_t addr, char *buf, size_t size); pid_t (*p_getpid)(void *prochandle); int (*p_obj_iter)(void *prochandle, tnfctl_ind_obj_f *func, void *client_data); } tnfctl_ind_config_t;</pre> The first field <i>p_read</i> is the address of a function that can read <i>size</i> bytes at address <i>addr</i> in the target image into the buffer <i>buf</i>. The function should return 0 upon success.. The second field <i>p_write</i> is the address of a function that can write <i>size</i> bytes at address <i>addr</i> in the target image from the buffer <i>buf</i>. The function should return 0 upon success. The third field <i>p_getpid</i> is the address of a function that should return the process id of the target process (<i>prochandle</i>). The fourth field <i>p_obj_iter</i> is the address of a function that iterates over all load objects and the executable by calling the callback function <i>func</i> with <i>client_data</i>. If <i>func</i> returns 0, <i>p_obj_iter</i> should continue processing link objects. If <i>func</i> returns any other value, <i>p_obj_iter</i> should stop calling the callback function and return that value. <i>p_obj_iter</i> should return 0 if it iterates over all load objects. If a failure is returned by any of the functions in <i>config</i>, the error is propagated back as <code>PREX_ERR_INTERNAL</code> by the <code>libtnfctl</code> interface that called it. The definition of <code>tnfctl_ind_obj_f</code> is:

tnfctl_indirect_open(3TNF)

```
typedef int
tnfctl_ind_obj_f(void *prochandle,
    const struct tnfctl_ind_obj_info *obj
    void *client_data);
typedef struct tnfctl_ind_obj_info {
    int      objfd;          /* -1 indicates fd not available */
    paddr_t text_base;      /* virtual addr of text segment */
    paddr_t data_base;      /* virtual addr of data segment */
    const char *objname;     /* null-term. pathname to loadobj */
} tnfctl_ind_obj_info_t;
```

objfd should be the file descriptor of the load object or executable. If it is `-1`, then *objname* should be an absolute pathname to the load object or executable. If *objfd* is not closed by `libtnfctl`, it should be closed by the load object iterator function. *text_base* and *data_base* are the addresses where the text and data segments of the load object are mapped in the target process.

Whenever the target process opens or closes a dynamic object, the set of available probes may change. See `dlopen(3DL)` and `dlclose(3DL)`. In indirect mode, call `tnfctl_check_libs()` when such events occur to make `libtnfctl` aware of any changes. In other modes this is unnecessary but harmless. It is also harmless to call `tnfctl_check_libs()` when no such events have occurred.

RETURN VALUES `tnfctl_indirect_open()` and `tnfctl_check_libs()` return `TNFCTL_ERR_NONE` upon success.

ERRORS The following error codes apply to `tnfctl_indirect_open()`:

<code>TNFCTL_ERR_ALLOCFAIL</code>	A memory allocation failure occurred.
<code>TNFCTL_ERR_BUSY</code>	Internal tracing is being used.
<code>TNFCTL_ERR_NOLIBTNFPROBE</code>	<code>libtnfprobe.so.1</code> is not loaded in the target process.
<code>TNFCTL_ERR_INTERNAL</code>	An internal error occurred.

The following error codes apply to `tnfctl_check_libs()`:

<code>TNFCTL_ERR_ALLOCFAIL</code>	A memory allocation failure occurred.
<code>TNFCTL_ERR_INTERNAL</code>	An internal error occurred.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO `prex(1)`, `TNF_PROBE(3TNF)`, `dlclose(3DL)`, `dlopen(3DL)`, `libtnfctl(3TNF)`, `tnfctl_probe_enable(3TNF)`, `tnfctl_probe_trace(3TNF)`, `tracing(3TNF)`, `proc(4)`, `attributes(5)`

tnfctl_indirect_open(3TNF)

Linker and Libraries Guide

NOTES tnfctl_indirect_open() should only be called after the dynamic linker has mapped in all the libraries (rtld sync point) and called only after the process is stopped. Indirect process probe control assumes the target process is stopped whenever any libtnfctl interface is used on it. For example, when used for indirect process probe control, tnfctl_probe_enable(3TNF) and tnfctl_probe_trace(3TNF) should be called only for a process that is stopped.

tnfctl_internal_open(3TNF)

NAME	tnfctl_internal_open – create handle for internal process probe control								
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_internal_open(tnfctl_handle_t **ret_val) ;</pre>								
DESCRIPTION	tnfctl_internal_open() returns in <i>ret_val</i> a pointer to an opaque handle that can be used to control probes in the same process as the caller (internal process probe control). The process must have libtnfprobe.so.1 loaded. Probes in libraries that are brought in by dlopen(3DL) will be visible after the library has been opened. Probes in libraries closed by a dlclose(3DL) will not be visible after the library has been disassociated. See the NOTES section for more details.								
RETURN VALUES	tnfctl_internal_open() returns TNFCTL_ERR_NONE upon success.								
ERRORS	<table> <tr> <td>TNFCTL_ERR_ALLOCFAIL</td><td>A memory allocation failure occurred.</td></tr> <tr> <td>TNFCTL_ERR_BUSY</td><td>Another client is already tracing this program (internally or externally).</td></tr> <tr> <td>TNFCTL_ERR_NOLIBTNFPROBE</td><td>libtnfprobe.so.1 is not linked in the target process.</td></tr> <tr> <td>TNFCTL_ERR_INTERNAL</td><td>An internal error occurred.</td></tr> </table>	TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.	TNFCTL_ERR_BUSY	Another client is already tracing this program (internally or externally).	TNFCTL_ERR_NOLIBTNFPROBE	libtnfprobe.so.1 is not linked in the target process.	TNFCTL_ERR_INTERNAL	An internal error occurred.
TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.								
TNFCTL_ERR_BUSY	Another client is already tracing this program (internally or externally).								
TNFCTL_ERR_NOLIBTNFPROBE	libtnfprobe.so.1 is not linked in the target process.								
TNFCTL_ERR_INTERNAL	An internal error occurred.								
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:								
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtnfc</td></tr> <tr> <td>MT Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfc	MT Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtnfc								
MT Level	MT-Safe								
SEE ALSO	ld(1), prex(1), TNF_PROBE(3TNF), dlopen(3DL), dlclose(3DL), libtnfctl(3TNF), tracing(3TNF), attributes(5) <i>Linker and Libraries Guide</i>								
NOTES	libtnfctl interposes on dlopen(3DL) and dlclose(3DL) in order to be notified of libraries being dynamically opened and closed. This interposition is necessary for internal process probe control to update its list of probes. In these interposition functions, a lock is acquired to synchronize on traversal of the library list maintained by the runtime linker. To avoid deadlocking on this lock, tnfctl_internal_open() should not be called from within the init section of a library that can be opened by dlopen(3DL).								

`tnfctl_internal_open(3TNF)`

Since interposition does not work as expected when a library is opened dynamically, `tnfctl_internal_open()` should not be used if the client opened `libtnfctl` through `dlopen(3DL)`. In this case, the client program should be built with a static dependency on `libtnfctl`. Also, if the client program is explicitly linking in `-ldl`, it should link `-ltnfctl` before `-ldl`.

Probes in filtered libraries (see `ld(1)`) will not be seen because the filtee (backing library) is loaded lazily on the first symbol reference and not at process startup or `dlopen(3DL)` time. A workaround is to call `tnfctl_check_libs(3TNF)` once the caller is sure that the filtee has been loaded.

tnfctl_kernel_open(3TNF)

NAME	tnfctl_kernel_open – create handle for kernel probe control										
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_kernel_open(tnfctl_handle_t **ret_val);</pre>										
DESCRIPTION	tnfctl_kernel_open() starts a kernel tracing session and returns in ret_val an opaque handle that can be used to control tracing and probes in the kernel. Only one kernel tracing session is possible at a time on a given machine. An error code of TNFCTL_ERR_BUSY is returned if there is another process using kernel tracing. Use the command <pre>fuser -f /dev/tnfctl</pre> to print the process id of the process currently using kernel tracing. Only a superuser may use tnfctl_kernel_open(). An error code of TNFCTL_ERR_ACCES is returned if the caller does not have the necessary privileges.										
RETURN VALUES	tnfctl_kernel_open returns TNFCTL_ERR_NONE upon success.										
ERRORS	<table> <tr> <td>TNFCTL_ERR_ACCES</td><td>Permission denied. Superuser privileges are needed for kernel tracing.</td></tr> <tr> <td>TNFCTL_ERR_BUSY</td><td>Another client is currently using kernel tracing.</td></tr> <tr> <td>TNFCTL_ERR_ALLOCFAIL</td><td>Memory allocation failed.</td></tr> <tr> <td>TNFCTL_ERR_FILENOTFOUND</td><td>/dev/tnfctl not found.</td></tr> <tr> <td>TNFCTL_ERR_INTERNAL</td><td>Some other failure occurred.</td></tr> </table>	TNFCTL_ERR_ACCES	Permission denied. Superuser privileges are needed for kernel tracing.	TNFCTL_ERR_BUSY	Another client is currently using kernel tracing.	TNFCTL_ERR_ALLOCFAIL	Memory allocation failed.	TNFCTL_ERR_FILENOTFOUND	/dev/tnfctl not found.	TNFCTL_ERR_INTERNAL	Some other failure occurred.
TNFCTL_ERR_ACCES	Permission denied. Superuser privileges are needed for kernel tracing.										
TNFCTL_ERR_BUSY	Another client is currently using kernel tracing.										
TNFCTL_ERR_ALLOCFAIL	Memory allocation failed.										
TNFCTL_ERR_FILENOTFOUND	/dev/tnfctl not found.										
TNFCTL_ERR_INTERNAL	Some other failure occurred.										
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtnfc</td></tr> <tr> <td>MT Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfc	MT Level	MT-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Availability	SUNWtnfc										
MT Level	MT-Safe										
SEE ALSO	prex(1), fuser(1M), TNF_PROBE(3TNF), libtnfctl(3TNF), tracing(3TNF), tnf_kernel_probes(4), attributes(5)										

NAME	tnfctl_pid_open, tnfctl_exec_open, tnfctl_continue – interfaces for direct probe and process control for another process
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_pid_open(pid_t pid, tnfctl_handle_t **ret_val) ; tnfctl_errcode_t tnfctl_exec_open(const char *pgm_name, char * const *argv, char * const *envp, const char *libnfp_path, const char *ld_preload, tnfctl_handle_t **ret_val) ; tnfctl_errcode_t tnfctl_continue(tnfctl_handle_t *hndl, tnfctl_event_t *evt, tnfctl_handle_t **child_hndl) ;</pre>
DESCRIPTION	The <code>tnfctl_pid_open()</code>, <code>tnfctl_exec_open()</code>, and <code>tnfctl_continue()</code> functions create handles to control probes in another process (direct process probe control). Either <code>tnfctl_pid_open()</code> or <code>tnfctl_exec_open()</code> will return a handle in <code>ret_val</code> that can be used for probe control. On return of these calls, the process is stopped. <code>tnfctl_continue()</code> allows the process specified by <code>hndl</code> to continue execution. The <code>tnfctl_pid_open()</code> function attaches to a running process with process id of <code>pid</code>. The process is stopped on return of this call. The <code>tnfctl_pid_open()</code> function returns an error message if <code>pid</code> is the same as the calling process. See <code>tnfctl_internal_open(3TNF)</code> for information on internal process probe control. A pointer to an opaque handle is returned in <code>ret_val</code>, which can be used to control the process and the probes in the process. The target process must have <code>libtnfprobe.so.1</code> (defined in <code><tnf/tnfctl.h></code> as macro <code>TNFCTL_LIBTNFPROBE</code>) linked in for probe control to work. The <code>tnfctl_exec_open()</code> function is used to <code>exec(2)</code> a program and obtain a probe control handle. For probe control to work, the process image to be <code>exec'd</code> must load <code>libtnfprobe.so.1</code>. The <code>tnfctl_exec_open()</code> function makes it simple for the library to be loaded at process start up time. The <code>pgm_name</code> argument is the command to <code>exec</code>. If <code>pgm_name</code> is not an absolute path, then the <code>\$PATH</code> environment variable is used to find the <code>pgm_name</code>. <code>argv</code> is a null-terminated argument pointer, that is, it is a null-terminated array of pointers to null-terminated strings. These strings constitute the argument list available to the new process image. The <code>argv</code> argument must have at least one member, and it should point to a string that is the same as <code>pgm_name</code>. See <code>execve(2)</code>. The <code>libnfp_path</code> argument is an optional argument, and if set, it should be the path to the directory that contains <code>libtnfprobe.so.1</code>. There is no need for a trailing <code>"/"</code> in this argument. This argument is useful if <code>libtnfprobe.so.1</code> is not installed in <code>/usr/lib</code>. <code>ld_preload</code> is a space-separated list of libraries to preload into the target program. This string should follow the syntax guidelines of the <code>LD_PRELOAD</code> environment variable. See <code>ld.so.1(1)</code>. The following illustrates how strings are concatenated to form the <code>LD_PRELOAD</code> environment variable in the new process image:

tnfctl_pid_open(3TNF)

```
<current value of $LD_PRELOAD> + <space> +  
libtnfprobe_path + "/libtnfprobe.so.1" +<space> +  
ld_preload
```

This option is useful for preloading interposition libraries that have probes in them.

envp is an optional argument, and if set, it is used for the environment of the target program. It is a null-terminated array of pointers to null-terminated strings. These strings constitute the environment of the new process image. See `execve(2)`. If *envp* is set, it overrides *ld_preload*. In this case, it is the caller's responsibility to ensure that `libtnfprobe.so.1` is loaded into the target program. If *envp* is not set, the new process image inherits the environment of the calling process, except for `LD_PRELOAD`.

The *ret_val* argument is the handle that can be used to control the process and the probes within the process. Upon return, the process is stopped before any user code, including `.init` sections, has been executed.

The `tnfctl_continue()` function is a blocking call and lets the target process referenced by *hndl* continue running. It can only be used on handles returned by `tnfctl_pid_open()` and `tnfctl_exec_open()` (direct process probe control). It returns when the target stops; the reason that the process stopped is returned in *evt*. This call is interruptible by signals. If it is interrupted, the process is stopped, and `TNFCTL_EVENT_EINTR` is returned in *evt*. The client of this library will have to decide which signal implies a stop to the target and catch that signal. Since a signal interrupts `tnfctl_continue()`, it will return, and the caller can decide whether or not to call `tnfctl_continue()` again.

`tnfctl_continue()` returns with an event of `TNFCTL_EVENT_DLOPEN`, `TNFCTL_EVENT_DLCLOSE`, `TNFCTL_EVENT_EXEC`, `TNFCTL_EVENT_FORK`, `TNFCTL_EVENT_EXIT`, or `TNFCTL_EVENT_TARGGONE`, respectively, when the target program calls `dlopen(3DL)`, `dlclose(3DL)`, any flavor of `exec(2)`, `fork(2)` (or `fork1(2)`), `exit(2)`, or terminates unexpectedly. If the target program called `exec(2)`, the client then needs to call `tnfctl_close(3TNF)` on the current handle leaving the target resumed, suspended, or killed (second argument to `tnfctl_close(3TNF)`). No other `libtnfctl` interface call can be used on the existing handle. If the client wants to control the `exec`'ed image, it should leave the old handle suspended, and use `tnfctl_pid_open()` to reattach to the same process. This new handle can then be used to control the `exec`'ed image. See **EXAMPLES** below for sample code. If the target process did a `fork(2)` or `fork1(2)`, and if control of the child process is not needed, then *child_hndl* should be `NULL`. If control of the child process is needed, then *child_hndl* should be set. If it is set, a pointer to a handle that can be used to control the child process is returned in *child_hndl*. The child process is stopped at the end of the `fork()` system call. See **EXAMPLES** for an example of this event.

RETURN VALUES

The `tnfctl_pid_open()`, `tnfctl_exec_open()`, and `tnfctl_continue()` functions return `TNFCTL_ERR_NONE` upon success.

ERRORS

The following error codes apply to `tnfctl_pid_open()`:

tnfctl_pid_open(3TNF)

TNFCTL_ERR_BADARG	The <i>pid</i> specified is the same process. Use <code>tnfctl_internal_open(3TNF)</code> instead.
TNFCTL_ERR_ACCES	Permission denied. No privilege to connect to a setuid process.
TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.
TNFCTL_ERR_BUSY	Another client is already using <code>/proc</code> to control this process or internal tracing is being used.
TNFCTL_ERR_NOTDYNAMIC	The process is not a dynamic executable.
TNFCTL_ERR_NOPROCESS	No such target process exists.
TNFCTL_ERR_NOLIBTNFPROBE	<code>libtnfprobe.so.1</code> is not linked in the target process.
TNFCTL_ERR_INTERNAL	An internal error occurred.

The following error codes apply to `tnfctl_exec_open()`:

TNFCTL_ERR_ACCES	Permission denied.
TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.
TNFCTL_ERR_NOTDYNAMIC	The target is not a dynamic executable.
TNFCTL_ERR_NOLIBTNFPROBE	<code>libtnfprobe.so.1</code> is not linked in the target process.
TNFCTL_ERR_FILENOTFOUND	The program is not found.
TNFCTL_ERR_INTERNAL	An internal error occurred.

The following error codes apply to `tnfctl_continue()`:

TNFCTL_ERR_BADARG	Bad input argument. <i>hndl</i> is not a direct process probe control handle.
TNFCTL_ERR_INTERNAL	An internal error occurred.
TNFCTL_ERR_NOPROCESS	No such target process exists.

EXAMPLES **EXAMPLE 1** Using `tnfctl_pid_open()`

These examples do not include any error-handling code. Only the initial example includes the declaration of the variables that are used in all of the examples.

The following example shows how to preload `libtnfprobe.so.1` from the normal location and inherit the parent's environment.

```
const char      *pgm;
char * const    *argv;
tnfctl_handle_t *hndl, *new_hndl, *child_hndl;
tnfctl_errcode_t err;
```

tnfctl_pid_open(3TNF)

EXAMPLE 1 Using tnfctl_pid_open() (Continued)

```
char * const      *envp_ptr;
extern char        **environ;
tnfctl_event_t     evt;
int                pid;

/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, NULL, NULL, &hndl);
```

This example shows how to preload two user-supplied libraries `libc_probe.so.1` and `libthread_probe.so.1`. They interpose on the corresponding `libc.so` and `libthread.so` interfaces and have probes for function entry and exit. `libtnfprobe.so.1` is preloaded from the normal location and the parent's environment is inherited.

```
/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, NULL,
    "libc_probe.so.1 libthread_probe.so.1", &hndl);
```

This example preloads an interposition library `libc_probe.so.1`, and specifies a different location from which to preload `libtnfprobe.so.1`.

```
/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, "/opt/SUNWXXX/lib",
    "libc_probe.so.1", &hndl);
```

To set up the environment explicitly for probe control to work, the target process must link `libtnfprobe.so.1`. If using *envp*, it is the caller's responsibility to do so.

```
/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
/* envp_ptr set up to caller's needs */
err = tnfctl_exec_open(pgm, argv, envp_ptr, NULL, NULL, &hndl);
```

Use this example to resume a process that does an `exec(2)` without controlling it.

```
err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
case TNFCTL_EVENT_EXEC:
    /* let target process continue without control */
    err = tnfctl_close(hndl, TNFCTL_TARG_RESUME);
    ...
    break;
}
```

Alternatively, use the next example to control a process that does an `exec(2)`.

EXAMPLE 1 Using tnfctl_pid_open() (Continued)

```
/*
 * assume the pid variable has been set by calling
 * tnfctl_trace_attrs_get()
 */
err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
case TNFCTL_EVENT_EXEC:
    /* suspend the target process */
    err = tnfctl_close(hndl, TNFCTL_TARG_SUSPEND);
    /* re-open the exec'ed image */
    err = tnfctl_pid_open(pid, &new_hndl);
    /* new_hndl now controls the exec'ed image */
    ...
    break;
}
```

To let fork'ed children continue without control, use NULL as the last argument to tnfctl_continue().

```
err = tnfctl_continue(hndl, &evt, NULL);
```

The next example is how to control child processes that fork(2) or fork1(2) create.

```
err = tnfctl_continue(hndl, &evt, &child_hndl);
switch (evt) {
case TNFCTL_EVENT_FORK:
    /* spawn a new thread or process to control child_hndl */
    ...
    break;
}
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO ld(1), prex(1), proc(1), exec(2), execve(2), exit(2), fork(2), TNF_PROBE(3TNF), dlclose(3DL), dlopen(3DL), libtnfctl(3TNF), tnfctl_close(3TNF), tnfctl_internal_open(3TNF), tracing(3TNF) attributes(5)

Linker and Libraries Guide

NOTES After a call to tnfctl_continue() returns, a client should use tnfctl_trace_attrs_get(3TNF) to check the trace_buf_state member of the trace attributes and make sure that there is no internal error in the target.

tnfctl_probe_apply(3TNF)

NAME	tnfctl_probe_apply, tnfctl_probe_apply_ids – iterate over probes				
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_probe_apply(tnfctl_handle_t *hndl, tnfctl_probe_op_t probe_op, void *clientdata); tnfctl_errcode_t tnfctl_probe_apply_ids(tnfctl_handle_t *hndl, ulong_t probe_count, ulong_t *probe_ids, tnfctl_probe_op_t probe_op, void *clientdata);</pre>				
DESCRIPTION	tnfctl_probe_apply() is used to iterate over the probes controlled by <i>hndl</i>. For every probe, the <i>probe_op</i> function is called: <pre>typedef tnfctl_errcode_t (*tnfctl_probe_op_t)(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *clientdata);</pre> Several predefined functions are available for use as <i>probe_op</i>. These functions are described in tnfctl_probe_state_get(3TNF). The <i>clientdata</i> supplied in tnfctl_probe_apply() is passed in as the last argument of <i>probe_op</i>. The <i>probe_hndl</i> in the probe operation function can be used to query or change the state of the probe. See tnfctl_probe_state_get(3TNF). The <i>probe_op</i> function should return TNFCTL_ERR_NONE upon success. It can also return an error code, which will cause tnfctl_probe_apply() to stop processing the rest of the probes and return with the same error code. Note that there are five (5) error codes reserved that the client can use for its own semantics. See ERRORS. The lifetime of <i>probe_hndl</i> is the same as the lifetime of <i>hndl</i>. It is good until <i>hndl</i> is closed by tnfctl_close(3TNF). Do not confuse a <i>probe_hndl</i> with <i>hndl</i>. The <i>probe_hndl</i> refers to a particular probe, while <i>hndl</i> refers to a process or the kernel. If <i>probe_hndl</i> is used in another libtnfctl(3TNF) interface, and it references a probe in a library that has been dynamically closed (see dlclose(3DL)), then the error code TNFCTL_ERR_INVALIDPROBE will be returned by that interface. tnfctl_probe_apply_ids() is very similar to tnfctl_probe_apply(). The difference is that <i>probe_op</i> is called only for probes that match a probe id specified in the array of integers referenced by <i>probe_ids</i>. The number of probe ids in the array should be specified in <i>probe_count</i>. Use tnfctl_probe_state_get() to get the <i>probe_id</i> that corresponds to the <i>probe_hndl</i>.				
RETURN VALUES	tnfctl_probe_apply() and tnfctl_probe_apply_ids() return TNFCTL_ERR_NONE upon success.				
ERRORS	The following errors apply to both tnfctl_probe_apply() and tnfctl_probe_apply_ids(): <table> <tr> <td>TNFCTL_ERR_INTERNAL</td><td>An internal error occurred.</td></tr> <tr> <td>TNFCTL_ERR_USR1</td><td>Error code reserved for user.</td></tr> </table>	TNFCTL_ERR_INTERNAL	An internal error occurred.	TNFCTL_ERR_USR1	Error code reserved for user.
TNFCTL_ERR_INTERNAL	An internal error occurred.				
TNFCTL_ERR_USR1	Error code reserved for user.				

tnfctl_probe_apply(3TNF)

TNFCTL_ERR_USR2	Error code reserved for user.
TNFCTL_ERR_USR3	Error code reserved for user.
TNFCTL_ERR_USR4	Error code reserved for user.
TNFCTL_ERR_USR5	Error code reserved for user.

tnfctl_probe_apply() and tnfctl_probe_apply_ids() also return any error returned by the callback function *probe_op*.

The following errors apply only to tnfctl_probe_apply_ids():

TNFCTL_ERR_INVALIDPROBE	The probe handle is no longer valid. For example, the probe is in a library that has been closed by dlclose(3DL).

EXAMPLES

EXAMPLE 1 Enabling Probes

To enable all probes:

```
tnfctl_probe_apply(hndl, tnfctl_probe_enable, NULL);
```

EXAMPLE 2 Disabling Probes

To disable the probes that match a certain pattern in the probe attribute string:

```
/* To disable all probes that contain the string "vm" */
tnfctl_probe_apply(hndl, select_disable, "vm");
static tnfctl_errcode_t
select_disable(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl,
void *client_data)
{
    char *pattern = client_data;
    tnfctl_probe_state_t probe_state;
    tnfctl_probe_state_get(hndl, probe_hndl, &probe_state);
    if (strstr(probe_state.attr_string, pattern)) {
        tnfctl_probe_disable(hndl, probe_hndl, NULL);
    }
}
```

Note that these examples do not have any error handling code.

ATTRIBUTES

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

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

tnfctl_probe_apply(3TNF)

SEE ALSO | prex(1), TNF_PROBE(3TNF), dlclose(3DL), dlopen(3DL), libtnfctl(3TNF),
tnfctl_close(3TNF), tnfctl_probe_state_get(3TNF), tracing(3TNF),
tnf_kernel_probes(4), attributes(5)

Linker and Libraries Guide

tnfctl_probe_state_get(3TNF)

NAME	tnfctl_probe_state_get, tnfctl_probe_enable, tnfctl_probe_disable, tnfctl_probe_trace, tnfctl_probe_untrace, tnfctl_probe_connect, tnfctl_probe_disconnect_all – interfaces to query and to change the state of a probe						
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_probe_state_get(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, tnfctl_probe_state_t *state); tnfctl_errcode_t tnfctl_probe_enable(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *ignored); tnfctl_errcode_t tnfctl_probe_disable(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *ignored); tnfctl_errcode_t tnfctl_probe_trace(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *ignored); tnfctl_errcode_t tnfctl_probe_untrace(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *ignored); tnfctl_errcode_t tnfctl_probe_disconnect_all(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, void *ignored); tnfctl_errcode_t tnfctl_probe_connect(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl, const char *lib_base_name, const char *func_name);</pre>						
DESCRIPTION	tnfctl_probe_state_get() returns the state of the probe specified by <i>probe_hndl</i> in the process or kernel specified by <i>hndl</i>. The user will pass these in to an apply iterator. The caller must also allocate <i>state</i> and pass in a pointer to it. The semantics of the individual members of <i>state</i> are: <table> <tr> <td>id</td><td>The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls <i>tnfctl_apply()</i>, <i>tnfctl_apply_ids()</i>, or <i>tnfctl_register_funcs()</i>.</td></tr> <tr> <td>attr_string</td><td>A string that consists of <i>attribute value</i> pairs separated by semicolons. For the syntax of this string, see the syntax of the detail argument of the TNF_PROBE(3TNF) macro. The attributes <i>name</i>, <i>slots</i>, <i>keys</i>, <i>file</i>, and <i>line</i> are defined for every probe. Additional user-defined attributes can be added by using the <i>detail</i> argument of the TNF_PROBE(3TNF) macro. An example of <i>attr_string</i> follows: <pre>"name pageout;slots vnode pages_pageout ; keys vm pageio io;file vm.c;line 25;"</pre> </td></tr> <tr> <td>enabled</td><td>B_TRUE if the probe is enabled, or B_FALSE if the probe is disabled. Probes are disabled by default. Use</td></tr> </table>	id	The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls <i>tnfctl_apply()</i> , <i>tnfctl_apply_ids()</i> , or <i>tnfctl_register_funcs()</i> .	attr_string	A string that consists of <i>attribute value</i> pairs separated by semicolons. For the syntax of this string, see the syntax of the detail argument of the TNF_PROBE(3TNF) macro. The attributes <i>name</i> , <i>slots</i> , <i>keys</i> , <i>file</i> , and <i>line</i> are defined for every probe. Additional user-defined attributes can be added by using the <i>detail</i> argument of the TNF_PROBE(3TNF) macro. An example of <i>attr_string</i> follows: <pre>"name pageout;slots vnode pages_pageout ; keys vm pageio io;file vm.c;line 25;"</pre>	enabled	B_TRUE if the probe is enabled, or B_FALSE if the probe is disabled. Probes are disabled by default. Use
id	The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls <i>tnfctl_apply()</i> , <i>tnfctl_apply_ids()</i> , or <i>tnfctl_register_funcs()</i> .						
attr_string	A string that consists of <i>attribute value</i> pairs separated by semicolons. For the syntax of this string, see the syntax of the detail argument of the TNF_PROBE(3TNF) macro. The attributes <i>name</i> , <i>slots</i> , <i>keys</i> , <i>file</i> , and <i>line</i> are defined for every probe. Additional user-defined attributes can be added by using the <i>detail</i> argument of the TNF_PROBE(3TNF) macro. An example of <i>attr_string</i> follows: <pre>"name pageout;slots vnode pages_pageout ; keys vm pageio io;file vm.c;line 25;"</pre>						
enabled	B_TRUE if the probe is enabled, or B_FALSE if the probe is disabled. Probes are disabled by default. Use						

tnfctl_probe_state_get(3TNF)

	tnfctl_probe_enable() or tnfctl_probe_disable() to change this state.
traced	B_TRUE if the probe is traced, or B_FALSE if the probe is not traced. Probes in user processes are traced by default. Kernel probes are untraced by default. Use tnfctl_probe_trace() or tnfctl_probe_untrace() to change this state.
new_probe	B_TRUE if this is a new probe brought in since the last change in libraries. See dlopen(3DL) or dlclose(3DL). Otherwise, the value of new_probe will be B_FALSE. This field is not meaningful for kernel probe control.
obj_name	The name of the shared object or executable in which the probe is located. This string can be freed, so the client should make a copy of the string if it needs to be saved for use by other libtnfctl interfaces. In kernel mode, this string is always NULL.
func_names	A null-terminated array of pointers to strings that contain the names of functions connected to this probe. Whenever an enabled probe is encountered at runtime, these functions are executed. This array also will be freed by the library when the state of the probe changes. Use tnfctl_probe_connect() or tnfctl_probe_disconnect_all() to change this state.
func_addrs	A null-terminated array of pointers to addresses of functions in the target image connected to this probe. This array also will be freed by the library when the state of the probe changes.
client_registered_data	Data that was registered by the client for this probe by the creator function in tnfctl_register_funcs(3TNF).
tnfctl_probe_enable(), tnfctl_probe_disable(), tnfctl_probe_trace(), tnfctl_probe_untrace(), and tnfctl_probe_disconnect_all() ignore the last argument. This convenient feature permits these functions to be used in the <i>probe_op</i> field of tnfctl_probe_apply(3TNF) and tnfctl_probe_apply_ids(3TNF). tnfctl_probe_enable() enables the probe specified by <i>probe_hdl</i> . This is the master switch on a probe. A probe does not perform any action until it is enabled. tnfctl_probe_disable() disables the probe specified by <i>probe_hdl</i> .	

tnfctl_probe_state_get(3TNF)

tnfctl_probe_trace() turns on tracing for the probe specified by *probe_hdl*. Probes emit a trace record only if the probe is traced.

tnfctl_probe_untrace() turns off tracing for the probe specified by *probe_hdl*. This is useful if you want to connect probe functions to a probe without tracing it.

tnfctl_probe_connect() connects the function *func_name* which exists in the library *lib_base_name*, to the probe specified by *probe_hdl*. tnfctl_probe_connect() returns an error code if used on a kernel tnfctl handle. *lib_base_name* is the base name (not a path) of the library. If it is NULL, and multiple functions in the target process match *func_name*, one of the matching functions is chosen arbitrarily. A probe function is a function that is in the target's address space and is written to a certain specification. The specification is not currently published.

tnf_probe_debug() is one function exported by libtnfprobe.so.1 and is the debug function that prex(1) uses. When the debug function is executed, it prints out the probe arguments and the value of the sunw%debug attribute of the probe to stderr.

tnfctl_probe_disconnect_all() disconnects all probe functions from the probe specified by *probe_hdl*.

Note that no libtnfctl call returns a probe handle (tnfctl_probe_t), yet each of the routines described here takes a *probe_hdl* as an argument. These routines may be used by passing them to one of the tnfctl_probe_apply(3TNF) iterators as the "op" argument. Alternatively, probe handles may be obtained and saved by a user's "op" function, and they can be passed later as the *probe_hdl* argument when using any of the functions described here.

RETURN VALUES

tnfctl_probe_state_get(), tnfctl_probe_enable(), tnfctl_probe_disable(), tnfctl_probe_trace(), tnfctl_probe_untrace(), tnfctl_probe_disconnect_all() and tnfctl_probe_connect() return TNFCTL_ERR_NONE upon success.

ERRORS

The following error codes apply to tnfctl_probe_state_get():

TNFCTL_ERR_INVALIDPROBE	<i>probe_hdl</i> is no longer valid. The library that the probe was in could have been dynamically closed by dlclos(3DL).

The following error codes apply to tnfctl_probe_enable(), tnfctl_probe_disable(), tnfctl_probe_trace(), tnfctl_probe_untrace(), and tnfctl_probe_disconnect_all()

TNFCTL_ERR_INVALIDPROBE	<i>probe_hdl</i> is no longer valid. The library that the probe was in could have been dynamically closed by dlclos(3DL).
TNFCTL_ERR_BUFBROKEN	Cannot do probe operations because tracing is broken in the target.

tnfctl_probe_state_get(3TNF)

TNFCTL_ERR_NOBUF Cannot do probe operations until a buffer is allocated. See tnfctl_buffer_alloc(3TNF). This error code does not apply to kernel probe control.

The following error codes apply to tnfctl_probe_connect():

TNFCTL_ERR_INVALIDPROBE *probe_hdl* is no longer valid. The library that the probe was in could have been dynamically closed by dlclose(3DL).

TNFCTL_ERR_BADARG The handle is a kernel handle, or *func_name* could not be found.

TNFCTL_ERR_BUFBROKEN Cannot do probe operations because tracing is broken in the target.

TNFCTL_ERR_NOBUF Cannot do probe operations until a buffer is allocated. See tnfctl_buffer_alloc(3TNF).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO prex(1), TNF_PROBE(3TNF), libtnfctl(3TNF), tnfctl_check_libs(3TNF), tnfctl_continue(3TNF), tnfctl_probe_apply(3TNF), tnfctl_probe_apply_ids(3TNF), tracing(3TNF), tnf_kernel_probes(4), attributes(5)

tnfctl_register_funcs(3TNF)

NAME	tnfctl_register_funcs – register callbacks for probe creation and destruction						
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_register_funcs(tnfctl_handle_t *hndl, void * (*create_func) (tnfctl_handle_t *, tnfctl_probe_t *), void (*destroy_func) (void *));</pre>						
DESCRIPTION	The function <code>tnfctl_register_funcs()</code> is used to store client-specific data on a per-probe basis. It registers a creator and a destructor function with <i>hndl</i>, either of which can be NULL. The creator function is called for every probe that currently exists in <i>hndl</i>. Every time a new probe is discovered, that is brought in by <code>dlopen(3DL)</code>, <i>create_func</i> is called. The return value of the creator function is stored as part of the probe state and can be retrieved by <code>tnfctl_probe_state_get(3TNF)</code> in the member field <i>client_registered_data</i>. <i>destroy_func</i> is called for every probe handle that is freed. This does not necessarily happen at the time <code>dlclose(3DL)</code> frees the shared object. The probe handles are freed only when <i>hndl</i> is closed by <code>tnfctl_close(3TNF)</code>. If <code>tnfctl_register_funcs()</code> is called a second time for the same <i>hndl</i>, then the previously registered destructor function is called first for all of the probes.						
RETURN VALUES	<code>tnfctl_register_funcs()</code> returns <code>TNFCTL_ERR_NONE</code> upon success.						
ERRORS	<code>TNFCTL_ERR_INTERNAL</code> An internal error occurred.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:						
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWtnfc</td></tr><tr><td>MT Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfc	MT Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtnfc						
MT Level	MT-Safe						
SEE ALSO	<code>prex(1)</code>, <code>TNF_PROBE(3TNF)</code>, <code>dlclose(3DL)</code>, <code>dlopen(3DL)</code>, <code>libtnfctl(3TNF)</code>, <code>tnfctl_close(3TNF)</code>, <code>tnfctl_probe_state_get(3TNF)</code>, <code>tracing(3TNF)</code>, <code>tnf_kernel_probes(4)</code>, <code>attributes(5)</code> <i>Linker and Libraries Guide</i>						

tnfctl_strerror(3TNF)

NAME	tnfctl_strerror – map a tnfctl error code to a string						
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> const char * tnfctl_strerror(tnfctl_errcode_t <i>errcode</i>);</pre>						
DESCRIPTION	tnfctl_strerror() maps the error number in <i>errcode</i> to an error message string, and it returns a pointer to that string. The returned string should not be overwritten or freed.						
ERRORS	tnfctl_strerror() returns the string "unknown libtnfctl.so error code" if the error number is not within the legal range.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>Availability</td><td>SUNWtnfc</td></tr><tr><td>MT Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfc	MT Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtnfc						
MT Level	MT-Safe						
SEE ALSO	prex(1), TNF_PROBE(3TNF), libtnfctl(3TNF), tracing(3TNF), attributes(5)						

NAME	tnfctl_trace_attrs_get – get the trace attributes from a tnfctl handle										
SYNOPSIS	<pre>cc [flag...] file... -ltnfctl [library...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_trace_attrs_get(tnfctl_handle_t *hndl, tnfctl_trace_attrs_t *attrs);</pre>										
DESCRIPTION	The <code>tnfctl_trace_attrs_get()</code> function returns the trace attributes associated with <i>hndl</i> in <i>attrs</i>. The trace attributes can be changed by some of the other interfaces in <code>libtnfctl(3TNF)</code>. It is the client's responsibility to use <code>tnfctl_trace_attrs_get()</code> to get the new trace attributes after use of interfaces that change them. Typically, a client will use <code>tnfctl_trace_attrs_get()</code> after a call to <code>tnfctl_continue(3TNF)</code> in order to make sure that tracing is still working. See the discussion of <code>trace_buf_state</code> that follows. Trace attributes are represented by the struct <code>tnfctl_trace_attrs</code> structure defined in <code><tnf/tnfctl.h></code>: <pre>struct tnfctl_trace_attrs { pid_t targ_pid; /* not kernel mode */ const char *trace_file_name; /* not kernel mode */ size_t trace_buf_size; size_t trace_min_size; tnfctl_bufstate_t trace_buf_state; boolean_t trace_state; boolean_t filter_state; /* kernel mode only */ long pad; };</pre> The semantics of the individual members of <i>attrs</i> are: <table> <tr> <td><code>targ_pid</code></td><td>The process id of the target process. This is not valid for kernel tracing.</td></tr> <tr> <td><code>trace_file_name</code></td><td>The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces.</td></tr> <tr> <td><code>trace_buf_size</code></td><td>The size of the trace buffer or file in bytes.</td></tr> <tr> <td><code>trace_min_size</code></td><td>The minimum size in bytes of the trace buffer that can be allocated by using the <code>tnfctl_buffer_alloc(3TNF)</code> interface.</td></tr> <tr> <td><code>trace_buf_state</code></td><td>The state of the trace buffer. <code>TNFCTL_BUF_OK</code> indicates that a trace buffer has been allocated. <code>TNFCTL_BUF_NONE</code> indicates that no buffer has been allocated. <code>TNFCTL_BUF_BROKEN</code> indicates that there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will</td></tr> </table>	<code>targ_pid</code>	The process id of the target process. This is not valid for kernel tracing.	<code>trace_file_name</code>	The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces.	<code>trace_buf_size</code>	The size of the trace buffer or file in bytes.	<code>trace_min_size</code>	The minimum size in bytes of the trace buffer that can be allocated by using the <code>tnfctl_buffer_alloc(3TNF)</code> interface.	<code>trace_buf_state</code>	The state of the trace buffer. <code>TNFCTL_BUF_OK</code> indicates that a trace buffer has been allocated. <code>TNFCTL_BUF_NONE</code> indicates that no buffer has been allocated. <code>TNFCTL_BUF_BROKEN</code> indicates that there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will
<code>targ_pid</code>	The process id of the target process. This is not valid for kernel tracing.										
<code>trace_file_name</code>	The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces.										
<code>trace_buf_size</code>	The size of the trace buffer or file in bytes.										
<code>trace_min_size</code>	The minimum size in bytes of the trace buffer that can be allocated by using the <code>tnfctl_buffer_alloc(3TNF)</code> interface.										
<code>trace_buf_state</code>	The state of the trace buffer. <code>TNFCTL_BUF_OK</code> indicates that a trace buffer has been allocated. <code>TNFCTL_BUF_NONE</code> indicates that no buffer has been allocated. <code>TNFCTL_BUF_BROKEN</code> indicates that there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will										

tnfctl_trace_attrs_get(3TNF)

	be written. To fix tracing, restart the process. For kernel tracing, deallocate the existing buffer with <code>tnfctl_buffer_dealloc(3TNF)</code> and allocate a new one with <code>tnfctl_buffer_alloc(3TNF)</code> .
trace_state	The global tracing state of the target. Probes that are enabled will not write out data unless this state is on. This state is off by default for the kernel and can be changed by <code>tnfctl_trace_state_set(3TNF)</code> . For a process, this state is on by default and can only be changed by <code>tnf_process_disable(3TNF)</code> and <code>tnf_process_enable(3TNF)</code> .
filter_state	The state of process filtering. For kernel probe control, it is possible to select a set of processes for which probes are enabled. See <code>tnfctl_filter_list_get(3TNF)</code> , <code>tnfctl_filter_list_add(3TNF)</code> , and <code>tnfctl_filter_list_delete(3TNF)</code> . No trace output will be written when other processes traverse these probe points. By default process filtering is off, and all processes cause the generation of trace records when they hit an enabled probe. Use <code>tnfctl_filter_state_set(3TNF)</code> to change the filter state.

RETURN VALUES The `tnfctl_trace_attrs_get()` function returns `TNFCTL_ERR_NONE` upon success.

ERRORS The `tnfctl_trace_attrs_get()` function will fail if:

<code>TNFCTL_ERR_INTERNAL</code>	An internal error occurred.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO `prex(1)`, `TNF_PROBE(3TNF)`, `libtnfctl(3TNF)`, `tnfctl_buffer_alloc(3TNF)`, `tnfctl_continue(3TNF)`, `tnfctl_filter_list_get(3TNF)`, `tnf_process_disable(3TNF)`, `tracing(3TNF)`, `attributes(5)`

tnfctl_trace_state_set(3TNF)

NAME	tnfctl_trace_state_set, tnfctl_filter_state_set, tnfctl_filter_list_get, tnfctl_filter_list_add, tnfctl_filter_list_delete – control kernel tracing and process filtering
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_trace_state_set(tnfctl_handle_t *hdl, boolean_t trace_state); tnfctl_errcode_t tnfctl_filter_state_set(tnfctl_handle_t *hdl, boolean_t filter_state); tnfctl_errcode_t tnfctl_filter_list_get(tnfctl_handle_t *hdl, pid_t **pid_list, int *pid_count); tnfctl_errcode_t tnfctl_filter_list_add(tnfctl_handle_t *hdl, pid_t pid_to_add); tnfctl_errcode_t tnfctl_filter_list_delete(tnfctl_handle_t *hdl, pid_t pid_to_delete);</pre>
DESCRIPTION	The interfaces to control kernel tracing and process filtering are used only with kernel handles, handles created by <code>tnfctl_kernel_open(3TNF)</code>. These interfaces are used to change the tracing and filter states for kernel tracing. <code>tnfctl_trace_state_set()</code> sets the kernel global tracing state to "on" if <i>trace_state</i> is <code>B_TRUE</code>, or to "off" if <i>trace_state</i> is <code>B_FALSE</code>. For the kernel, <i>trace_state</i> is off by default. Probes that are enabled will not write out data unless this state is on. Use <code>tnfctl_trace_attrs_get(3TNF)</code> to retrieve the current tracing state. <code>tnfctl_filter_state_set()</code> sets the kernel process filtering state to "on" if <i>filter_state</i> is <code>B_TRUE</code>, or to "off" if <i>filter_state</i> is <code>B_FALSE</code>. <i>filter_state</i> is off by default. If it is on, only probe points encountered by processes in the process filter set by <code>tnfctl_filter_list_add()</code> will generate trace points. Use <code>tnfctl_trace_attrs_get(3TNF)</code> to retrieve the current process filtering state. <code>tnfctl_filter_list_get()</code> returns the process filter list as an array in <i>pid_list</i>. The count of elements in the process filter list is returned in <i>pid_count</i>. The caller should use <code>free(3C)</code> to free memory allocated for the array <i>pid_list</i>. <code>tnfctl_filter_list_add()</code> adds <i>pid_to_add</i> to the process filter list. The process filter list is maintained even when the process filtering state is off, but it has no effect unless the process filtering state is on. <code>tnfctl_filter_list_delete()</code> deletes <i>pid_to_delete</i> from the process filter list. It returns an error if the process does not exist or is not in the filter list.
RETURN VALUES	The interfaces <code>tnfctl_trace_state_set()</code> , <code>tnfctl_filter_state_set()</code> , <code>tnfctl_filter_list_add()</code> , <code>tnfctl_filter_list_delete()</code> , and <code>tnfctl_filter_list_get()</code> return <code>TNFCTL_ERR_NONE</code> upon success.
ERRORS	The following error codes apply to <code>tnfctl_trace_state_set</code> :

tnfctl_trace_state_set(3TNF)

TNFCTL_ERR_BADARG	The handle is not a kernel handle.
TNFCTL_ERR_NOBUF	Cannot turn on tracing without a buffer being allocated.
TNFCTL_ERR_BUFBROKEN	Tracing is broken in the target.
TNFCTL_ERR_INTERNAL	An internal error occurred.
The following error codes apply to tnfctl_filter_state_set:	
TNFCTL_ERR_BADARG	The handle is not a kernel handle.
TNFCTL_ERR_INTERNAL	An internal error occurred.
The following error codes apply to tnfctl_filter_list_add:	
TNFCTL_ERR_BADARG	The handle is not a kernel handle.
TNFCTL_ERR_NOPROCESS	No such process exists.
TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.
TNFCTL_ERR_INTERNAL	An internal error occurred.
The following error codes apply to tnfctl_filter_list_delete:	
TNFCTL_ERR_BADARG	The handle is not a kernel handle.
TNFCTL_ERR_NOPROCESS	No such process exists.
TNFCTL_ERR_INTERNAL	An internal error occurred.
The following error codes apply to tnfctl_filter_list_get:	
TNFCTL_ERR_BADARG	The handle is not a kernel handle.
TNFCTL_ERR_ALLOCFAIL	A memory allocation failure occurred.
TNFCTL_ERR_INTERNAL	An internal error occurred.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfc
MT Level	MT-Safe

SEE ALSO

prex(1), TNF_PROBE(3TNF), free(3C), libtnfctl(3TNF),
tnfctl_kernel_open(3TNF), tnfctl_trace_attrs_get (3TNF),
tracing(3TNF), tnf_kernel_probes(4), attributes(5)

TNF_DECLARE_RECORD(3TNF)

NAME	TNF_DECLARE_RECORD, TNF_DEFINE_RECORD_1, TNF_DEFINE_RECORD_2, TNF_DEFINE_RECORD_3, TNF_DEFINE_RECORD_4, TNF_DEFINE_RECORD_5 – TNF type extension interface for probes
SYNOPSIS	<pre>cc [flag ...] file ... [-ltnfprobe] [library ...] #include <tnf/probe.h> TNF_DECLARE_RECORD (c_type, tnf_type) ; TNF_DEFINE_RECORD_1 (c_type, tnf_type, tnf_member_type_1, c_member_name_1) ; TNF_DEFINE_RECORD_2 (c_type, tnf_type, tnf_member_type_1, c_member_name_1, tnf_member_type_2, c_member_name_2) ; TNF_DEFINE_RECORD_3 (c_type, tnf_type, tnf_member_type_1, c_member_name_1, tnf_member_type_2, c_member_name_2, tnf_member_type_3, c_member_name_3) ; TNF_DEFINE_RECORD_4 (c_type, tnf_type, tnf_member_type_1, c_member_name_1, tnf_member_type_2, c_member_name_2, tnf_member_type_3, c_member_name_3, tnf_member_type_4, c_member_name_4) ; TNF_DEFINE_RECORD_5 (c_type, tnf_type, tnf_member_type_1, c_member_name_1, tnf_member_type_2, c_member_name_2, tnf_member_type_3, c_member_name_3, tnf_member_type_4, c_member_name_4, tnf_member_type_5, c_member_name_5) ;</pre>
DESCRIPTION	This macro interface is used to extend the TNF (Trace Normal Form) types that can be used in TNF_PROBE(3TNF). There should be only one TNF_DECLARE_RECORD and one TNF_DEFINE_RECORD per new type being defined. The TNF_DECLARE_RECORD should precede the TNF_DEFINE_RECORD. It can be in a header file that multiple source files share if those source files need to use the <i>tnf_type</i> being defined. The TNF_DEFINE_RECORD should only appear in one of the source files. The TNF_DEFINE_RECORD macro interface defines a function as well as a couple of data structures. Hence, this interface has to be used in a source file (.c or .cc file) at file scope and not inside a function. Note that there is no semicolon after the TNF_DEFINE_RECORD interface. Having one will generate a compiler warning. Compiling with the preprocessor option -DNPROBE (see cc(1B)), or with the preprocessor control statement #define NPROBE ahead of the #include <tnf/probe.h> statement, will stop the TNF type extension code from being compiled into the program. The <i>c_type</i> argument must be a C struct type. It is the template from which the new <i>tnf_type</i> is being created. Not all elements of the C struct need be provided in the TNF type being defined.

TNF_DECLARE_RECORD(3TNF)

The *tnf_type* argument is the name being given to the newly created type. Use of this interface uses the name space prefixed by *tnf_type*. If a new type called "xxx_type" is defined by a library, then the library should not use "xxx_type" as a prefix in any other symbols it defines. The policy on managing the type name space is the same as managing any other name space in a library; that is, prefix any new TNF types by the unique prefix that the rest of the symbols in the library use. This would prevent name space collisions when linking multiple libraries that define new TNF types. For example, if a library `libpalloc.so` uses the prefix "pal" for all symbols it defines, then it should also use the prefix "pal" for all new TNF types being defined.

The *tnf_member_type_n* argument is the TNF type of the *n*th provided member of the C structure.

The *tnf_member_name_n* argument is the name of the *n*th provided member of the C structure.

EXAMPLES

EXAMPLE 1 Defining and using a TNF type.

The following example demonstrates how a new TNF type is defined and used in a probe. This code is assumed to be part of a fictitious library called "libpalloc.so" which uses the prefix "pal" for all its symbols.

```
#include <tnf/probe.h>
typedef struct pal_header {
    long    size;
    char *  descriptor;
    struct pal_header *next;
} pal_header_t;
TNF_DECLARE_RECORD(pal_header_t, pal_tnf_header);
TNF_DEFINE_RECORD_2(pal_header_t, pal_tnf_header,
                    tnf_long,    size,
                    tnf_string, descriptor)

/*
 * Note: name space prefixed by pal_tnf_header should not be used by this
 *       client anymore.
 */
void
pal_free(pal_header_t *header_p)
{
    int state;
    TNF_PROBE_2(pal_free_start, "palloc pal_free",
               "sunw%debug entering pal_free",
               tnf_long,    state_var, state,
               pal_tnf_header, header_var, header_p);
    . . .
}
```

ATTRIBUTES

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

TNF_DECLARE_RECORD(3TNF)

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

SEE ALSO `prex(1)`, `tnfdump(1)`, `TNF_PROBE(3TNF)`, `tnf_process_disable(3TNF)`, `attributes(5)`

NOTES It is possible to make a *tnf_type* definition be recursive or mutually recursive e.g. a structure that uses the "next" field to point to itself (a linked list). If such a structure is sent in to a `TNF_PROBE(3TNF)`, then the entire linked list will be logged to the trace file (until the "next" field is NULL). But, if the list is circular, it will result in an infinite loop. To break the recursion, either don't include the "next" field in the *tnf_type*, or define the type of the "next" member as `tnf_opaque`.

TNF_PROBE(3TNF)

NAME	TNF_PROBE, TNF_PROBE_0, TNF_PROBE_1, TNF_PROBE_2, TNF_PROBE_3, TNF_PROBE_4, TNF_PROBE_5, TNF_PROBE_0_DEBUG, TNF_PROBE_1_DEBUG, TNF_PROBE_2_DEBUG, TNF_PROBE_3_DEBUG, TNF_PROBE_4_DEBUG, TNF_PROBE_5_DEBUG, TNF_DEBUG – probe insertion interface
SYNOPSIS	<pre>cc [flag ...] [-DTNF_DEBUG] file ... [-ltnfprobe] [library ...] #include <tnf/probe.h></pre> TNF_PROBE_0(name, keys, detail); TNF_PROBE_1(name, keys, detail, arg_type_1, arg_name_1, arg_value_1); TNF_PROBE_2(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2); TNF_PROBE_3(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3); TNF_PROBE_4(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4); TNF_PROBE_5(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4, arg_type_5, arg_name_5, arg_value_5); TNF_PROBE_0_DEBUG(name, keys, detail); TNF_PROBE_1_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1); TNF_PROBE_2_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2); TNF_PROBE_3_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3); TNF_PROBE_4_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4); TNF_PROBE_5_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4, arg_type_5, arg_name_5, arg_value_5);
DESCRIPTION	This macro interface is used to insert probes into C or C++ code for tracing. See <code>tracing(3TNF)</code> for a discussion of the Solaris tracing architecture, including example source code that uses it. You can place probes anywhere in C and C++ programs including <code>.init</code> sections, <code>.fini</code> sections, multi-threaded code, shared objects, and shared objects opened by <code>dlopen(3DL)</code>. Use probes to generate trace data for performance analysis or to write debugging output to <code>stderr</code>. Probes are controlled at runtime by <code>prex(1)</code>.

The trace data is logged to a trace file in Trace Normal Form (TNF). The interface for the user to specify the name and size of the trace file is described in `prex(1)`. Think of the trace file as the least recently used circular buffer. Once the file has been filled, newer events will overwrite the older ones.

Use `TNF_PROBE_0` through `TNF_PROBE_5` to create production probes. These probes are compiled in by default. Developers are encouraged to embed such probes strategically, and to leave them compiled within production software. Such probes facilitate on-site analysis of the software.

Use `TNF_PROBE_0_DEBUG` through `TNF_PROBE_5_DEBUG` to create debug probes. These probes are compiled out by default. If you compile the program with the preprocessor option `-DTNF_DEBUG` (see `cc(1B)`), or with the preprocessor control statement `#define TNF_DEBUG` ahead of the `#include <tnf/probe.h>` statement, the debug probes will be compiled into the program. When compiled in, debug probes differ in only one way from the equivalent production probes. They contain an additional "debug" attribute which may be used to distinguish them from production probes at runtime, for example, when using `prex()`. Developers are encouraged to embed any number of probes for debugging purposes. Disabled probes have such a small runtime overhead that even large numbers of them do not make a significant impact.

If you compile with the preprocessor option `-DNPROBE` (see `cc(1B)`), or place the preprocessor control statement `#define NPROBE` ahead of the `#include <tnf/probe.h>` statement, no probes will be compiled into the program.

name	The <i>name</i> of the probe should follow the syntax guidelines for identifiers in ANSI C. The use of <i>name</i> declares it, hence no separate declaration is necessary. This is a block scope declaration, so it does not affect the name space of the program.
keys	<i>keys</i> is a string of space-separated keywords that specify the groups that the probe belongs to. Semicolons, single quotation marks, and the equal character (=) are not allowed in this string. If any of the groups are enabled, the probe is enabled. <i>keys</i> cannot be a variable. It must be a string constant.
detail	<i>detail</i> is a string that consists of <attribute> <value> pairs that are each separated by a semicolon. The first word (up to the space) is considered to be the attribute and the rest of the string (up to the semicolon) is considered the value. Single quotation marks are used to denote a string value. Besides quotation marks, spaces separate multiple values. The value is optional. Although semicolons or single quotation marks generally are not allowed within either the attribute or the value, when text with embedded spaces is meant to denote a single value, use single quotes surrounding this text. Use <i>detail</i> for one of two reasons. First, use <i>detail</i> to supply an attribute that a user can type into <code>prex(1)</code> to select probes. For example, if a user defines an attribute called <i>color</i> , then <code>prex(1)</code> can select probes based on the value of <i>color</i> . Second, use <i>detail</i> to

TNF_PROBE(3TNF)

annotate a probe with a string that is written out to a trace file only once. `prex(1)` uses spaces to tokenize the value when searching for a match. Spaces around the semicolon delimiter are allowed. *detail* cannot be a variable; it must be a string constant. For example, the *detail* string:

```
"XYZ%debug 'entering function A'; XYZ%exception 'no file';
XYZ%func_entry; XYZ%color red blue"
```

consists of 4 units:

Attribute	Value	Values that prex matches on
XYZ%debug	'entering function A'	'entering function A'
XYZ%exception	'no file'	'no file'
XYZ%func_entry	/.*/	(regular expression)
XYZ%color	red blue	red <or> blue

Attribute names must be prefixed by the vendor stock symbol followed by the '%' character. This avoids conflicts in the attribute name space. All attributes that do not have a '%' character are reserved. The following attributes are predefined:

Attribute	Semantics
name	name of probe
keys	keys of the probe (value is space-separated tokens)
file	file name of the probe
line	line number of the probe
slots	slot names of the probe event (<i>arg_name_n</i>)
object	the executable or shared object that this probe is in.
debug	distinguishes debug probes from production probes

arg_type_n

This is the type of the *n*th argument. The following are predefined TNF types:

tnf Type	Associated C type (and semantics)
tnf_int	int

TNF_PROBE(3TNF)

tnf Type	Associated C type (and semantics)
tnf_uint	unsigned int
tnf_long	long
tnf_ulong	unsigned long
tnf_longlong	long long (if implemented in compilation system)
tnf_ulonglong	unsigned long long (if implemented in compilation system)
tnf_float	float
tnf_double	double
tnf_string	char *
tnf_opaque	void *

To define new TNF types that are records consisting of the predefined TNF types or references to other user defined types, use the interface specified in `TNF_DECLARE_RECORD(3TNF)`.

arg_name_n *arg_name_n* is the name that the user associates with the *n*th argument. Do not place quotation marks around *arg_name_n*. Follow the syntax guidelines for identifiers in ANSI C. The string version of *arg_name_n* is stored for every probe and can be accessed as the attribute "slots".

arg_value_n *arg_value_n* is evaluated to yield a value to be included in the trace file. A read access is done on any variables that are mentioned in *arg_value_n*. In a multi-threaded program, it is the user's responsibility to place locks around the `TNF_PROBE` macro if *arg_value_n* contains a variable that should be read protected.

EXAMPLES **EXAMPLE 1** `tracing(3TNF)`.

See `tracing(3TNF)` for complete examples showing debug and production probes in source code.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfd
MT Level	MT-Safe

TNF_PROBE(3TNF)

SEE ALSO `cc(1B)`, `ld(1)`, `prex(1)`, `tnfdump(1)`, `dlopen(3DL)`, `libtnfctl(3TNF)`, `TNF_DECLARE_RECORD(3TNF)`, `threads(3THR)`, `tnf_process_disable(3TNF)`, `tracing(3TNF)`, `attributes(5)`

NOTES If attaching to a running program with `prex(1)` to control the probes, compile the program with `-lttnfprobe` or start the program with the environment variable `LD_PRELOAD` set to `libtnfprobe.so.1`. See `ld(1)`. If `libtnfprobe` is explicitly linked into the program, it must be before `libthread` on the link line.

NAME	tnf_process_disable, tnf_process_enable, tnf_thread_disable, tnf_thread_enable – probe control internal interface								
SYNOPSIS	<pre>cc [flag ...] file ... -ltnfprobe [library ...] #include <tnf/probe.h> void tnf_process_disable(void); void tnf_process_enable(void); void tnf_thread_disable(void); void tnf_thread_enable(void);</pre>								
DESCRIPTION	There are three levels of granularity for controlling tracing and probe functions (called probing from here on): probing for the entire process, a particular thread, and the probe itself can be disabled or enabled. The first two (process and thread) are controlled by this interface. The probe is controlled with the <code>prex(1)</code> utility. The <code>tnf_process_disable()</code> function turns off probing for the process. The default process state is to have probing enabled. The <code>tnf_process_enable()</code> function turns on probing for the process. The <code>tnf_thread_disable()</code> function turns off probing for the currently running thread. Threads are "born" or created with this state enabled. The <code>tnf_thread_enable()</code> function turns on probing for the currently running thread. If the program is a non-threaded program, these two thread interfaces disable or enable probing for the process.								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtnfd</td></tr> <tr> <td>Interface Stability</td><td>Unstable</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtnfd	Interface Stability	Unstable	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtnfd								
Interface Stability	Unstable								
MT-Level	MT-Safe								
SEE ALSO	<code>prex(1)</code> , <code>tnfdump(1)</code> , <code>TNF_DECLARE_RECORD(3TNF)</code> , <code>TNF_PROBE(3TNF)</code> , <code>attributes(5)</code>								
NOTES	A probe is considered enabled only if: ■ <code>prex(1)</code> has enabled the probe AND ■ the process has probing enabled, which is the default or could be set with <code>tnf_process_enable()</code> AND ■ the thread that hits the probe has probing enabled, which is every thread's default or could be set with <code>tnf_thread_enable()</code>.								

`tnf_process_disable(3TNF)`

There is a run time cost associated with determining that the probe is disabled. To reduce the performance effect of probes, this cost should be minimized. The quickest way that a probe can be determined to be disabled is by the enable control that `prex(1)` uses. Therefore, to disable all the probes in a process use the `disable` command in `prex(1)` rather than `tnf_process_disable()`.

The `tnf_process_disable()` and `tnf_process_enable()` functions should only be used to toggle probing based on some internal program condition. The `tnf_thread_disable()` function should be used to turn off probing for threads that are uninteresting.

NAME	tracing – overview of tnf tracing system
DESCRIPTION	tnf tracing is a set of programs and API's that can be used to present a high-level view of the performance of an executable, a library, or part of the kernel. tracing is used to analyze a program's performance and identify the conditions that produced a bug. The core elements of tracing are: TNF_PROBE_* () The TNF_PROBE_* () macros define "probes" to be placed in code which, when enabled and executed, cause information to be added to a trace file. See TNF_PROBE(3TNF). If there are insufficient TNF_PROBE_* macros to store all the data of interest for a probe, data may be grouped into records. See TNF_DECLARE_RECORD(3TNF). prex Displays and controls probes in running software. See prex(1). kernel probes A set of probes built into the Solaris kernel which capture information about system calls, multithreading, page faults, swapping, memory management, and I/O. You can use these probes to obtain detailed traces of kernel activity under your application workloads. See tnf_kernel_probes(4). tnfextract A program that extracts the trace data from the kernel's in-memory buffer into a file. See tnfextract(1). tnfdump A program that displays the information from a trace file. See tnfdump(1). libtnfctl A library of interfaces that controls probes in a process. See libtnfctl(3TNF). prex(1) also utilizes this library. Other tools and processes use the libtnfctl interfaces to exercise fine control over their own probes. tnf_process_enable () A routine called by a process to turn on tracing and probe functions for the current process. See tnf_process_enable(3TNF). tnf_process_disable () A routine called by a process to turn off tracing and probe functions for the current process. See tnf_process_disable(3TNF). tnf_thread_enable () A routine called by a process to turn on tracing and probe functions for the currently running thread. See tnf_thread_enable(3TNF).

tracing(3TNF)

`tnf_thread_disable()` A routine called by a process to turn off tracing and probe functions for the currently running thread. See `tnf_thread_disable(3TNF)`.

EXAMPLES

EXAMPLE 1 Tracing a Process

The following function in some daemon process accepts job requests of various types, queueing them for later execution. There are two "debug probes" and one "production probe." Note that probes which are intended for debugging will not be compiled into the final version of the code; however, production probes are compiled into the final product.

```
/*
 * To compile in all probes (for development):
 *   cc -DTNF_DEBUG ...
 *
 * To compile in only production probes (for release):
 *   cc ...
 *
 * To compile in no probes at all:
 *   cc -DNPROBE ...
 */
#include <tnf/probe.h>
void work(long, char *);
enum work_request_type { READ, WRITE, ERASE, UPDATE };
static char *work_request_name[] = {"read", "write", "erase", "update"};
main( )
{
    long i;
    for (i = READ; i <= UPDATE; i++)
        work(i, work_request_name[i]);
}
void work(long request_type, char *request_name)
{
    static long q_length;
    TNF_PROBE_2_DEBUG(work_start, "work",
        "XYZ%debug 'in function work'",
        tnf_long, request_type_arg, request_type,
        tnf_string, request_name_arg, request_name);
    /* assume work request is queued for later processing */
    q_length++;
    TNF_PROBE_1(work_queue, "work queue",
        "XYZ%work_load heavy",
        tnf_long, queue_length, q_length);
    TNF_PROBE_0_DEBUG(work_end, "work", "");
}
```

The production probe "work_queue," which remains compiled in the code, will, when enabled, log the length of the work queue each time a request is received.

The debug probes "work_start" and "work_end," which are compiled only during the development phase, track entry to and exit from the `work()` function and measure how much time is spent executing it. Additionally, the debug probe "work_start" logs

EXAMPLE 1 Tracing a Process (Continued)

the value of the two incoming arguments `request_type` and `request_name`. The runtime overhead for disabled probes is low enough that one can liberally embed them in the code with little impact on performance.

For debugging, the developer would compile with `-DTNF_DEBUG`, run the program under control of `prex(1)`, enable the probes of interest (in this case, all probes), continue the program until exit, and dump the trace file:

```
% cc
-DTNF_DEBUG -o daemon daemon.c # compile in all probes
% prex daemon                  # run program under prex control
Target process stopped
Type "continue" to resume the target, "help" for help ...
prex> list probes $all          # list all probes in program
<probe list output here>
prex> enable $all               # enable all probes
prex> continue                  # let target process execute
<program output here>
prex: target process finished
% ls /tmp/trace-*               # trace output is in trace-<pid>
/tmp/trace-4194
% tnfdump /tmp/trace-4194       # get ascii output of trace file
<trace records output here>
```

For the production version of the system, the developer simply compiles without `-DTNF_DEBUG`.

EXAMPLE 2 Tracing the Kernel

Kernel tracing is similar to tracing a process; however, there are some differences. For instance, to trace the kernel, you need superuser privileges. The following example uses `prex(1)` and traces the probes in the kernel that capture system call information.

```
Allocate kernel
trace buffer and capture trace data:
root# prex -k
Type "help" for help ...
prex> buffer alloc 2m           # allocate kernel trace buffer
Buffer of size 2097152 bytes allocated
prex> list probes $all          # list all kernel probes
<probe list output here>
prex> list probes syscall       # list syscall probes
                                # (keys=syscall)
<syscall probes list output here>
prex> enable syscall            # enable only syscall probes
prex> ktrace on                 # turn on kernel tracing
<Run your application in another window at this point>
prex> ktrace off                # turn off kernel tracing
prex> quit                      # exit prex
Extract the kernel's trace buffer into a file:
root# tnfxtract /tmp/ktrace     # extract kernel trace buffer
Reset kernel tracing:
```

tracing(3TNF)

EXAMPLE 2 Tracing the Kernel (Continued)

```
root# prex -k
prex> disable $all          # disable all probes
prex> untrace $all          # untrace all probes
prex> buffer dealloc        # deallocate kernel trace buffer
prex> quit
```

CAUTION: Do not deallocate the trace buffer until you have extracted it into a trace file. Otherwise, you will lose the trace data that you collected from your experiment!

Examine the kernel trace file:

```
root# tnfdump /tmp/ktrace    # get ascii dump of trace file
<trace records output here>
```

prex can also attach to a running process, list probes, and perform a variety of other tasks.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtnfd
MT Level	MT-Safe

SEE ALSO

prex(1), tnfdump(1), tnfxtract(1), TNF_DECLARE_RECORD(3TNF), TNF_PROBE(3TNF), libtnfctl(3TNF), tnf_process_disable(3TNF), tnf_kernel_probes(4), attributes(5)

NAME	volmgt_acquire – reserve removable media device				
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <sys/types.h> #include <volmgt.h> int volmgt_acquire(char *dev, char *id, int ovr, char **err, pid_t *pidp);</pre>				
DESCRIPTION	The <code>volmgt_acquire()</code> routine reserves the removable media device specified as <i>dev</i>. <code>volmgt_acquire()</code> operates in two different modes, depending on whether or not Volume Management is running. See <code>vold(1M)</code>. If Volume Management <i>is</i> running, <code>volmgt_acquire()</code> attempts to reserve the removable media device specified as <i>dev</i>. Specify <i>dev</i> as <i>either</i> a symbolic device name (for example, <code>floppy0</code>) or a physical device pathname (for example, <code>/vol/dsk/unnamed_floppy</code>). If Volume Management <i>is not</i> running, <code>volmgt_acquire()</code> requires callers to specify a physical device pathname for <i>dev</i>. Specifying <i>dev</i> as a symbolic device name is <i>not</i> acceptable. In this mode, <code>volmgt_acquire()</code> relies entirely on the major and minor numbers of the device to determine whether or not the device is reserved. If <i>dev</i> is free, <code>volmgt_acquire()</code> updates the internal device reservation database with the caller's process id (<i>pid</i>) and the specified id string. If <i>dev</i> is reserved by another process, the reservation attempt fails and <code>volmgt_acquire()</code>: ■ sets <code>errno</code> to <code>EBUSY</code> ■ fills the caller's id value in the array pointed to by <i>err</i> ■ fills in the <i>pid</i> to which the pointer <i>pidp</i> points with the <i>pid</i> of the process which holds the reservation, if the supplied <i>pidp</i> is non-zero If the override <i>ovr</i> is non-zero, the call overrides the device reservation.				
RETURN VALUES	Upon successful completion, <code>volmgt_acquire()</code> returns a non-zero value. Upon failure, <code>volmgt_acquire()</code> returns 0. If the return value is 0, and <code>errno</code> is set to <code>EBUSY</code>, the address pointed to by <i>err</i> contains the string that was specified as <i>id</i> (when the device was reserved by the process holding the reservation).				
ERRORS	The <code>volmgt_acquire()</code> routine fails if one or more of the following are true: <table border="0"> <tr> <td style="vertical-align: top;">EINVAL</td><td>One of the specified arguments is invalid or missing.</td></tr> <tr> <td style="vertical-align: top;">EBUSY</td><td><i>dev</i> is already reserved by another process (and <i>ovr</i> was not set to a non-zero value)</td></tr> </table>	EINVAL	One of the specified arguments is invalid or missing.	EBUSY	<i>dev</i> is already reserved by another process (and <i>ovr</i> was not set to a non-zero value)
EINVAL	One of the specified arguments is invalid or missing.				
EBUSY	<i>dev</i> is already reserved by another process (and <i>ovr</i> was not set to a non-zero value)				

volmgt_acquire(3VOLMGT)

EXAMPLES

EXAMPLE 1 Using volmgt_acquire()

In the following example, Volume Management is running and the first floppy drive is reserved, accessed and released.

```
#include <volmgt.h>
char *errp;
if (!volmgt_acquire("floppy0", "FileMgr", 0, NULL,
    &errp, NULL)) {
    /* handle error case */
    . . .
}
/* floppy acquired - now access it */
if (!volmgt_release("floppy0")) {
    /* handle error case */
    . . .
}
```

EXAMPLE 2 Using volmgt_acquire() To Override A Lock On Another Process

The following example shows how callers can override a lock on another process using volmgt_acquire().

```
char *errp, buf[20];
int override = 0;
pid_t pid;
if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp,
    &pid)) {
    if (errno == EBUSY) {
        (void) printf("override %s (pid=%ld)?\n",
            errp, pid); {
            (void) fgets(buf, 20, stdin);
            if (buf[0] == 'y') {
                override++;
            }
        }
    } else {
        /* handle other errors */
        . . .
    }
}
if (override) {
    if (!volmgt_acquire("floppy0", "FileMgr", 1,
        &errp, NULL)) {
        /* really give up this time! */
        . . .
    }
}
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

volmgt_acquire(3VOLMGT)

SEE ALSO vold(1M), free(3C), malloc(3C), volmgt_release(3VOLMGT), attributes(5)

NOTES When returning a string through *err*, volmgt_acquire() allocates a memory area using malloc(3C). Use free(3C) to release the memory area when no longer needed.

The *ovr* argument is intended to allow callers to override the current device reservation. It is assumed that the calling application has determined that the current reservation can safely be cleared. See EXAMPLES.

volmgt_check(3VOLMGT)

NAME	volmgt_check – have Volume Management check for media				
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_check(char *pathname) ;</pre>				
DESCRIPTION	This routine asks Volume Management to check the specified <i>pathname</i> and determine if new media has been inserted in that drive. If a null pointer is passed in, then Volume Management will check each device it is managing that can be checked. If new media is found, volmgt_check() tells Volume Management to initiate any "actions" specified in /etc/vold.conf (see vold.conf(4)).				
RETURN VALUES	This routine returns 0 if no media was found, and a non-zero value if any media was found.				
ERRORS	This routine can fail, returning 0, if a stat(2) or open(2) of the supplied <i>pathname</i> fails, or if any of the following is true: <table><tr><td>ENXIO</td><td>Volume Management is not running.</td></tr><tr><td>EINTR</td><td>An interrupt signal was detected while checking for media.</td></tr></table>	ENXIO	Volume Management is not running.	EINTR	An interrupt signal was detected while checking for media.
ENXIO	Volume Management is not running.				
EINTR	An interrupt signal was detected while checking for media.				
EXAMPLES	EXAMPLE 1 Checking If Any New Media Is Inserted To check if any drive managed by Volume Management has any new media inserted in it: <pre>if (volmgt_check(NULL)) { (void) printf("Volume Management found media\n"); }</pre> This would also request Volume Management to take whatever action was specified in /etc/vold.conf for any media found.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>MT-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	cc(1B), volcheck(1), vold(1M), open(2), stat(2), volmgt_inuse(3VOLMGT), volmgt_running(3VOLMGT), vold.conf(4), attributes(5), volfs(7FS)				
NOTES	Volume Management must be running for this routine to work.				

`volmgt_check(3VOLMGT)`

Since `volmgt_check()` returns 0 for two different cases (both when no media is found, and when an error occurs), it is up to the user to check *errno* to differentiate the two, and to ensure that Volume Management is running.

volmgt_feature_enabled(3VOLMGT)

NAME	volmgt_feature_enabled – check whether specific Volume Management features are enabled				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -l volmgt [<i>library</i> ...] #include <volmgt.h> int volmgt_feature_enabled(char *feat_str);</pre>				
DESCRIPTION	The volmgt_feature_enabled() routine checks whether specific Volume Management features are enabled. volmgt_feature_enabled() checks for the Volume Management features passed in to it by the <i>feat_str</i> parameter. Currently, the only supported feature string that volmgt_feature_enabled() checks for is floppy-summit-interfaces. The floppy-summit-interfaces feature string checks for the presence of the libvolmgt routines volmgt_acquire() and volmgt_release(). The list of features that volmgt_feature_enabled() checks for is expected to expand in the future.				
RETURN VALUES	0 is returned if the specified feature is not currently available. A non-zero value indicates that the specified feature is currently available.				
EXAMPLES	EXAMPLE 1 A sample of the volmgt_feature_enabled() function. In the following example, volmgt_feature_enabled() checks whether the floppy-summit-interfaces feature is enabled. <pre>if (volmgt_feature_enabled("floppy-summit-interfaces")) { (void) printf("Media Sharing Routines ARE present\n"); } else { (void) printf("Media Sharing Routines are NOT present\n"); }</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	volmgt_acquire(3VOLMGT), volmgt_release(3VOLMGT), attributes(5)				

NAME	volmgt_inuse – check whether or not Volume Management is managing a pathname				
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_inuse(char *pathname);</pre>				
DESCRIPTION	volmgt_inuse() checks whether Volume Management is managing the specified <i>pathname</i> .				
RETURN VALUES	A non-zero value is returned if Volume Management is managing the specified <i>pathname</i> , otherwise 0 is returned.				
ERRORS	This routine can fail, returning 0, if a stat(2) of the supplied <i>pathname</i> or an open(2) of /dev/volctl fails, or if any of the following is true: <table> <tr> <td>ENXIO</td><td>Volume Management is not running.</td></tr> <tr> <td>EINTR</td><td>An interrupt signal was detected while checking for the supplied <i>pathname</i> for use.</td></tr> </table>	ENXIO	Volume Management is not running.	EINTR	An interrupt signal was detected while checking for the supplied <i>pathname</i> for use.
ENXIO	Volume Management is not running.				
EINTR	An interrupt signal was detected while checking for the supplied <i>pathname</i> for use.				
EXAMPLES	EXAMPLE 1 Using volmgt_inuse() To see if Volume Management is managing the first floppy disk: <pre>if (volmgt_inuse("/dev/rdiskette0") != 0) { (void) printf("volmgt is managing diskette 0\n"); } else { (void) printf("volmgt is NOT managing diskette 0\n"); }</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	cc(1B), vold(1M), open(2), stat(2), errno(3C), volmgt_check(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), volfs(7FS)				
NOTES	This routine requires Volume Management to be running. Since volmgt_inuse() returns 0 for two different cases (both when a volume is not in use, and when an error occurs), it is up to the user to check errno to differentiate the two, and to ensure that Volume Management is running.				

volmgt_ownspath(3VOLMGT)

NAME	volmgt_ownspath – check Volume Management name space for path						
SYNOPSIS	<pre>cc [flag]... <i>file</i>... -lvolmgt [library]... #include <volmgt.h> int volmgt_ownspath(char *<i>path</i>) ;</pre>						
PARAMETERS	<i>path</i> A string containing the path.						
DESCRIPTION	The volmgt_ownspath() function checks to see if a given <i>path</i> is contained in the Volume Management name space. This is achieved by comparing the beginning of the supplied path name with the output from volmgt_root(3VOLMGT)						
RETURN VALUES	The volmgt_ownspath() function returns a non-zero value if <i>path</i> is owned by Volume Management. It returns 0 if <i>path</i> is not in its name space or Volume Management is not running.						
EXAMPLES	EXAMPLE 1 Using volmgt_ownspath() The following example first checks if Volume Management is running, then checks the Volume Management name space for <i>path</i>, and then returns the <i>id</i> for the piece of media. <pre>char *path; ... if (volmgt_running()) { if (volmgt_ownspath(path)) { (void) printf("id of %s is %lld\n", path, media_getid(path)); } }</pre>						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT Level</td><td>Safe</td></tr><tr><td>Commitment Level</td><td>Public</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT Level	Safe	Commitment Level	Public
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT Level	Safe						
Commitment Level	Public						
SEE ALSO	volmgt_root(3VOLMGT), volmgt_running(3VOLMGT), attributes(5)						

NAME	volmgt_release – release removable media device reservation						
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_release(char *dev);</pre>						
DESCRIPTION	The <code>volmgt_release()</code> routine releases the removable media device reservation specified as <i>dev</i>. See <code>volmgt_acquire(3VOLMGT)</code> for a description of <i>dev</i>. If <i>dev</i> is reserved by the caller, <code>volmgt_release()</code> updates the internal device reservation database to indicate that the device is no longer reserved. If the requested device is reserved by another process, the release attempt fails and <code>errno</code> is set to 0.						
RETURN VALUES	Upon successful completion, <code>volmgt_release</code> returns a non-zero value. Upon failure, 0 is returned.						
ERRORS	On failure, <code>volmgt_release()</code> returns 0, and sets <code>errno</code> for one of the following conditions: <code>EINVAL</code> <i>dev</i> was invalid or missing. <code>EBUSY</code> <i>dev</i> was not reserved by the caller.						
EXAMPLES	EXAMPLE 1 Using <code>volmgt_release()</code> In the following example, Volume Management is running, and the first floppy drive is reserved, accessed and released. <pre>#include <volmgt.h> char *errp; if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp, NULL)) { /* handle error case */ . . . } /* floppy acquired - now access it */ if (!volmgt_release("floppy0")) { /* handle error case */ . . . }</pre>						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> <tr> <td>Interface Stability</td><td>Stable</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe	Interface Stability	Stable
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	MT-Safe						
Interface Stability	Stable						
SEE ALSO	<code>vold(1M)</code> , <code>volmgt_acquire(3VOLMGT)</code> , <code>attributes(5)</code>						

volmgt_root(3VOLMGT)

NAME	volmgt_root – return the Volume Management root directory				
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> const char *volmgt_root(void);</pre>				
DESCRIPTION	The volmgt_root() function returns the current Volume Management root directory, which by default is /vol but can be configured to be in a different location.				
RETURN VALUES	The volmgt_root() function returns pointer to a static string containing the root directory for Volume Management.				
ERRORS	This function may fail if an open() of /dev/volctl fails. If this occurs a pointer to the default Volume Management root directory is returned.				
EXAMPLES	EXAMPLE 1 Finding the Volume Management root directory. To find out where the Volume Management root directory is: <pre>if ((path = volmgt_root()) != NULL) { (void) printf("Volume Management root dir=%s\n", path); } else { (void) printf("can't find Volume Management root dir\n"); }</pre>				
FILES	/vol default location for the Volume Management root directory				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	cc(1B), vold(1M), open(2), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), volfs(7FS)				
NOTES	This function returns the default root directory location even when Volume Management is not running.				

volmgt_running(3VOLMGT)

NAME	volmgt_running – return whether or not Volume Management is running				
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_running(void);</pre>				
DESCRIPTION	volmgt_running() tells whether or not Volume Management is running.				
RETURN VALUES	A non-zero value is returned if Volume Management is running, else 0 is returned.				
ERRORS	volmgt_running() will fail, returning 0, if a stat(2) or open(2) of /dev/volctl fails, or if any of the following is true: <table><tr><td>ENXIO</td><td>Volume Management is not running.</td></tr><tr><td>EINTR</td><td>An interrupt signal was detected while checking to see if Volume Management was running.</td></tr></table>	ENXIO	Volume Management is not running.	EINTR	An interrupt signal was detected while checking to see if Volume Management was running.
ENXIO	Volume Management is not running.				
EINTR	An interrupt signal was detected while checking to see if Volume Management was running.				
EXAMPLES	EXAMPLE 1 Using volmgt_running() To see if Volume Management is running: <pre>if (volmgt_running() != 0) { (void) printf("Volume Management is running\n"); } else { (void) printf("Volume Management is NOT running\n"); }</pre>				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>MT-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	cc(1B), vold(1M), open(2), stat(2), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), attributes(5), volfs(7FS)				
NOTES	Volume Management must be running for many of the Volume Management library routines to work.				

volmgt_symname(3VOLMGT)

NAME	volmgt_symname, volmgt_symdev – convert between Volume Management symbolic names, and the devices that correspond to them								
SYNOPSIS	<pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> char *volmgt_symname(char *pathname) ; char *volmgt_symdev(char *symname) ;</pre>								
DESCRIPTION	These two routines compliment each other, translating between Volume Management's symbolic name for a device, called a <i>symname</i>, and the <i>/dev pathname</i> for that same device. volmgt_symname() converts a supplied <i>/dev pathname</i> to a <i>symname</i>, Volume Management's idea of that device's symbolic name (see volfs(7FS) for a description of Volume Management symbolic names). volmgt_symdev() does the opposite conversion, converting between a <i>symname</i>, Volume Management's idea of a device's symbolic name for a volume, to the <i>/dev pathname</i> for that device.								
RETURN VALUES	volmgt_symname() returns the symbolic name for the device pathname supplied, and volmgt_symdev() returns the device pathname for the supplied symbolic name. These strings are allocated upon success, and therefore must be freed by the caller when they are no longer needed (see free(3C)).								
ERRORS	volmgt_symname() can fail, returning a null string pointer, if a stat(2) of the supplied pathname fails, or if an open(2) of <i>/dev/volctl</i> fails, or if any of the following is true: <table> <tr> <td>ENXIO</td><td>Volume Management is not running.</td></tr> <tr> <td>EINTR</td><td>An interrupt signal was detected while trying to convert the supplied <i>pathname</i> to a <i>symname</i>.</td></tr> </table> volmgt_symdev() can fail if an open(2) of <i>/dev/volctl</i> fails, or if any of the following is true: <table> <tr> <td>ENXIO</td><td>Volume Management is not running.</td></tr> <tr> <td>EINTR</td><td>An interrupt signal was detected while trying to convert the supplied <i>symname</i> to a <i>/dev pathname</i>.</td></tr> </table>	ENXIO	Volume Management is not running.	EINTR	An interrupt signal was detected while trying to convert the supplied <i>pathname</i> to a <i>symname</i> .	ENXIO	Volume Management is not running.	EINTR	An interrupt signal was detected while trying to convert the supplied <i>symname</i> to a <i>/dev pathname</i> .
ENXIO	Volume Management is not running.								
EINTR	An interrupt signal was detected while trying to convert the supplied <i>pathname</i> to a <i>symname</i> .								
ENXIO	Volume Management is not running.								
EINTR	An interrupt signal was detected while trying to convert the supplied <i>symname</i> to a <i>/dev pathname</i> .								
EXAMPLES	EXAMPLE 1 Testing Floppies The following tests how many floppies Volume Management currently sees in floppy drives (up to 10): <pre>for (i=0; i < 10; i++) { (void) sprintf(path, "floppy%d", i); if (volmgt_symdev(path) != NULL) { (void) printf("volume %s is in drive %d\n",</pre>								

EXAMPLE 1 Testing Floppies *(Continued)*

```

        path, i);
    }
}

```

EXAMPLE 2 Finding The Symbolic Name

This code finds out what symbolic name (if any) Volume Management has for /dev/rdisk/c0t6d0s2:

```

if ((nm = volmgt_symname("/dev/rdisk/c0t6d0s2")) == NULL) {
    (void) printf("path not managed\n");
} else {
    (void) printf("path managed as %s\n", nm);
}

```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO cc(1B), vold(1M), open(2), stat(2), free(3C), malloc(3C), volmgt_check(3VOLMGT), volmgt_inuse(3VOLMGT), volmgt_running(3VOLMGT), attributes(5), volfs(7FS)

NOTES These routines only work when Volume Management is running.

BUGS There should be a straightforward way to query Volume Management for a list of all media types it's managing, and how many of each type are being managed.

wsreg_add_child_component(3WSREG)

NAME	wsreg_add_child_component, wsreg_remove_child_component, wsreg_get_child_components – add or remove a child component
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_add_child_component(Wsreg_component *comp, const Wsreg_component *childComp); int wsreg_remove_child_component(Wsreg_component *comp, const Wsreg_component *childComp); Wsreg_component **wsreg_get_child_components(const Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_add_child_component()</code> function adds the component specified by <i>childComp</i> to the list of child components contained in the component specified by <i>comp</i>. The <code>wsreg_remove_child_component()</code> function removes the component specified by <i>childComp</i> from the list of child components contained in the component specified by <i>comp</i>. The <code>wsreg_get_child_components()</code> function returns the list of child components contained in the component specified by <i>comp</i>.
RETURN VALUES	The <code>wsreg_add_child_component()</code> function returns a non-zero value if the specified child component was successfully added; otherwise, 0 is returned. The <code>wsreg_remove_child_component()</code> function returns a non-zero value if the specified child component was successfully removed; otherwise, 0 is returned. The <code>wsreg_get_child_components()</code> function returns a null-terminated array of <code>Wsreg_component</code> pointers that represents the specified component's list of child components. If the specified component has no child components, <code>NULL</code> is returned. The resulting array must be released by the caller through a call to <code>wsreg_free_component_array()</code>. See <code>wsreg_create_component(3WSREG)</code>.
USAGE	The parent-child relationship between components in the product install registry is used to record a product's structure. Product structure is the arrangement of features and components that make up a product. The structure of installed products can be displayed with the <code>prodreg</code> GUI. The child component must be installed and registered before the parent component can be. The registration of a parent component that has child components results in each of the child components being updated to reflect their parent component. Read access to the product install registry is required in order to use these functions because these relationships are held with lightweight component references that can only be fully resolved using the registry contents.

`wsreg_add_child_component(3WSREG)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `prodreg(1M)`, `wsreg_can_access_registry(3WSREG)`,
`wsreg_create_component(3WSREG)`, `wsreg_initialize(3WSREG)`,
`wsreg_register(3WSREG)`, `wsreg_set_parent(3WSREG)`, `attributes(5)`

wsreg_add_compatible_version(3WSREG)

NAME	wsreg_add_compatible_version, wsreg_remove_compatible_version, wsreg_get_compatible_versions – add or remove a backward-compatible version				
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_add_compatible_version(Wsreg_component *comp, const char *version); int wsreg_remove_compatible_version(Wsreg_component *comp, const char *version); char **wsreg_get_compatible_versions(const Wsreg_component *comp);</pre>				
DESCRIPTION	The <code>wsreg_add_compatible_version()</code> function adds the version string specified by <i>version</i> to the list of backward-compatible versions contained in the component specified by <i>comp</i>. The <code>wsreg_remove_compatible_version()</code> function removes the version string specified by <i>version</i> from the list of backward-compatible versions contained in the component specified by <i>comp</i>. The <code>wsreg_get_compatible_versions()</code> function returns the list of backward-compatible versions contained in the component specified by <i>comp</i>.				
RETURN VALUES	The <code>wsreg_add_compatible_version()</code> function returns a non-zero value if the specified backward-compatible version was successfully added; otherwise, 0 is returned. The <code>wsreg_remove_compatible_version()</code> function returns a non-zero value if the specified backward-compatible version was successfully removed; otherwise, 0 is returned. The <code>wsreg_get_compatible_versions()</code> function returns a null-terminated array of char pointers that represents the specified component's list of backward-compatible versions. If the specified component has no such versions, NULL is returned. The resulting array and its contents must be released by the caller.				
USAGE	The list of backward compatible versions is used to allow components that are used by multiple products to upgrade successfully without compromising any of its dependent products. The installer that installs such an update can check the list of backward-compatible versions and look at what versions are required by all of the dependent components to ensure that the upgrade will not result in a broken product.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				

`wsreg_add_compatible_version(3WSREG)`

SEE ALSO `prodreg(1M)`, `wsreg_initialize(3WSREG)`, `wsreg_register(3WSREG)`,
`wsreg_set_version(3WSREG)`, `attributes(5)`

wsreg_add_dependent_component(3WSREG)

NAME	wsreg_add_dependent_component, wsreg_remove_dependent_component, wsreg_get_dependent_components – add or remove a dependent component
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_add_dependent_component(Wsreg_component *comp, const Wsreg_component *dependentComp); int wsreg_remove_dependent_component(Wsreg_component *comp, const Wsreg_component *dependentComp); Wsreg_component **wsreg_get_dependent_components(const Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_add_dependent_component()</code> function adds the component specified by <i>dependentComp</i> to the list of dependent components contained in the component specified by <i>comp</i>. The <code>wsreg_remove_dependent_component()</code> function removes the component specified by <i>dependentComp</i> from the list of dependent components contained in the component specified by <i>comp</i>. The <code>wsreg_get_dependent_components()</code> function returns the list of dependent components contained in the component specified by <i>comp</i>.
RETURN VALUES	The <code>wsreg_add_dependent_component()</code> function returns a non-zero value if the specified dependent component was successfully added; otherwise, 0 is returned. The <code>wsreg_remove_dependent_component()</code> function returns a non-zero value if the specified dependent component was successfully removed; otherwise, 0 is returned. The <code>wsreg_get_dependent_components()</code> function returns a null-terminated array of <code>Wsreg_component</code> pointers that represents the specified component's list of dependent components. If the specified component has no dependent components, <code>NULL</code> is returned. The resulting array must be released by the caller through a call to <code>wsreg_free_component_array()</code>. See <code>wsreg_create_component(3WSREG)</code>.
USAGE	The relationship between two components in which one must be installed for the other to be complete is a dependent/required relationship. The component that is required by the other component is the required component. The component that requires the other is the dependent component. The required component must be installed and registered before the dependent component can be. Uninstaller applications should check the registry before uninstalling and unregistering components so a successful uninstallation of one product will not result in another product being compromised. Read access to the product install registry is required to use these functions because these relationships are held with lightweight component references that can only be fully resolved using the registry contents.

`wsreg_add_dependent_component(3WSREG)`

The act of registering a component having required components results in the converse dependent relationships being established automatically.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `wsreg_add_required_component(3WSREG)`,
`wsreg_can_access_registry(3WSREG)`, `wsreg_create_component(3WSREG)`,
`wsreg_initialize(3WSREG)`, `wsreg_register(3WSREG)`, `attributes(5)`

wsreg_add_display_name(3WSREG)

NAME	wsreg_add_display_name, wsreg_remove_display_name, wsreg_get_display_name, wsreg_get_display_languages – add, remove, or return a localized display name
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_add_display_name(Wsreg_component *comp, const char *language, const char *display_name); int wsreg_remove_display_name(Wsreg_component *comp, const char *language); char *wsreg_get_display_name(const Wsreg_component *comp, const char *language); char **wsreg_get_display_languages(const Wsreg_component *comp);</pre>
DESCRIPTION	For each of these functions, the <i>comp</i> argument specifies the component on which these functions operate. The <i>language</i> argument is the ISO 639 language code identifying a particular display name associated with the specified component. The <code>wsreg_add_display_name()</code> function adds the display name specified by <i>display_name</i> to the component specified by <i>comp</i>. The <code>wsreg_remove_display_name()</code> function removes a display name from the component specified by <i>comp</i>. The <code>wsreg_get_display_name()</code> function returns a display name from the component specified by <i>comp</i>. The <code>wsreg_get_display_languages()</code> returns the ISO 639 language codes for which display names are available from the component specified by <i>comp</i>.
RETURN VALUES	The <code>wsreg_add_display_name()</code> function returns a non-zero value if the display name was set correctly; otherwise 0 is returned. The <code>wsreg_remove_display_name()</code> function returns a non-zero value if the display name was removed; otherwise 0 is returned. The <code>wsreg_get_display_name()</code> function returns the display name from the specified component if the component has a display name for the specified language code. Otherwise, NULL is returned. The caller must not free the resulting display name. The <code>wsreg_get_display_languages()</code> function returns a null-terminated array of ISO 639 language codes for which display names have been set into the specified component. If no display names have been set, NULL is returned. It is the caller's responsibility to release the resulting array, but not the contents of the array.

`wsreg_add_display_name(3WSREG)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `wsreg_initialize(3WSREG)`, `attributes(5)`

wsreg_add_required_component(3WSREG)

NAME	wsreg_add_required_component, wsreg_remove_required_component, wsreg_get_required_components – add or remove a required component
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_add_required_component(Wsreg_component *comp, const Wsreg_component *requiredComp); int wsreg_remove_required_component(Wsreg_component *comp, const Wsreg_component *requiredComp); Wsreg_component **wsreg_get_required_components(const Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_add_required_component()</code> function adds the component specified by <i>requiredComp</i> to the list of required components contained in the component specified by <i>comp</i>. The <code>wsreg_remove_required_component()</code> function removes the component specified by <i>requiredComp</i> from the list of required components contained in the component specified by <i>comp</i>. The <code>wsreg_get_required_components()</code> function returns the list of required components contained in the component specified by <i>comp</i>.
RETURN VALUES	The <code>wsreg_add_required_component()</code> function returns a non-zero value if the specified required component was successfully added. Otherwise, 0 is returned. The <code>wsreg_remove_required_component()</code> function returns a non-zero value if the specified required component was successfully removed. Otherwise, 0 is returned. The <code>wsreg_get_required_components()</code> function returns a null-terminated array of <code>Wsreg_component</code> pointers that represents the specified component's list of required components. If the specified component has no required components, <code>NULL</code> is returned. The resulting array must be released by the caller through a call to <code>wsreg_free_component_array()</code>. See <code>wsreg_create_component(3WSREG)</code>.
USAGE	The relationship between two components in which one must be installed for the other to be complete is a dependent/required relationship. The component that is required by the other component is the required component. The component that requires the other is the dependent component. The required component must be installed and registered before the dependent component can be. Uninstaller applications should check the registry before uninstalling and unregistering components so a successful uninstallation of one product will not result in another product being compromised. Read access to the product install registry is required in order to use these functions because these relationships are held with lightweight component references that can only be fully resolved using the registry contents.

`wsreg_add_required_component(3WSREG)`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `wsreg_add_dependent_component(3WSREG)`,
`wsreg_can_access_registry(3WSREG)`, `wsreg_create_component(3WSREG)`,
`wsreg_initialize(3WSREG)`, `wsreg_register(3WSREG)`, `attributes(5)`

wsreg_can_access_registry(3WSREG)

NAME	wsreg_can_access_registry – determine access to product install registry
SYNOPSIS	<pre>cc [flag...] file ...-lwsreg [library...] #include <fcntl.h> #include <wsreg.h> int wsreg_can_access_registry(int access_flag);</pre>
DESCRIPTION	The <code>wsreg_can_access_registry()</code> function is used to determine what access, if any, an application has to the product install registry. The <code>access_flag</code> argument can be one of the following: <code>O_RDONLY</code> Inquire about read only access to the registry. <code>O_RDWR</code> Inquire about modify (read and write) access to the registry.
RETURN VALUES	The <code>wsreg_can_access_registry()</code> function returns non-zero if the specified access level is permitted. A return value of 0 indicates the specified access level is not permitted.
EXAMPLES	EXAMPLE 1 Initialize the registry and determine if access to the registry is permitted. <pre>#include <fcntl.h> #include <wsreg.h> int main(int argc, char **argv) { int result; if (wsreg_initialize(WSREG_INIT_NORMAL, NULL)) { printf("conversion recommended, sufficient access denied\n"); } if (wsreg_can_access_registry(O_RDONLY)) { printf("registry read access granted\n"); } else { printf("registry read access denied\n"); } if (wsreg_can_access_registry(O_RDWR)) { printf("registry read/write access granted\n"); } else { printf("registry read/write access denied\n"); } }</pre>
USAGE	The <code>wsreg_initialize(3WSREG)</code> function must be called before calls to <code>wsreg_can_access_registry()</code> can be made.
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:

wsreg_can_access_registry(3WSREG)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO wsreg_initialize(3WSREG), attributes(5)

wsreg_clone_component(3WSREG)

NAME	wsreg_clone_component – clone a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> Wsreg_component *wsreg_clone_component(const Wsreg_component *comp) ;</pre>				
DESCRIPTION	The wsreg_clone_component() function clones the component specified by <i>comp</i> .				
RETURN VALUES	The wsreg_clone_component() returns a pointer to a component that is configured exactly the same as the component specified by <i>comp</i> .				
USAGE	The resulting component must be released through a call to wsreg_free_component() by the caller. See wsreg_create_component(3WSREG).				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	wsreg_create_component(3WSREG), wsreg_initialize(3WSREG), wsreg_get(3WSREG), attributes(5)				

wsreg_components_equal(3WSREG)

NAME	wsreg_components_equal – determine equality of two components				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_components_equal(const Wsreg_component *comp1, const Wsreg_component *comp2);</pre>				
DESCRIPTION	The <code>wsreg_components_equal()</code> function determines if the component specified by the <i>comp1</i> argument is equal to the component specified by the <i>comp2</i> argument. Equality is evaluated based only on the content of the two components, not the order in which data was set into the components.				
RETURN VALUES	The <code>wsreg_components_equal()</code> function returns a non-zero value if the component specified by the <i>comp1</i> argument is equal to the component specified by the <i>comp2</i> argument. Otherwise, 0 is returned.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_clone_component(3WSREG)</code> , <code>wsreg_create_component(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_create_component(3WSREG)

NAME	wsreg_create_component, wsreg_free_component, wsreg_free_component_array – create or release a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> Wsreg_component *wsreg_create_component(const char *uuid); void wsreg_free_component(Wsreg_component *comp); int wsreg_free_component_array(Wsreg_component **complist);</pre>				
DESCRIPTION	The <code>wsreg_create_component()</code> function allocates a new component and assigns the <code>uuid</code> (universal unique identifier) specified by <code>uuid</code> to the resulting component. The <code>wsreg_free_component()</code> function releases the memory associated with the component specified by <code>comp</code>. The <code>wsreg_free_component_array()</code> function frees the null-terminated array of component pointers specified by <code>complist</code>. This function can be used to free the results of a call to <code>wsreg_get_all()</code>. See <code>wsreg_get(3WSREG)</code>.				
RETURN VALUES	The <code>wsreg_create_component()</code> function returns a pointer to the newly allocated <code>Wsreg_component</code> structure. The <code>wsreg_free_component_array()</code> function returns a non-zero value if the specified <code>Wsreg_component</code> array was freed successfully. Otherwise, 0 is returned.				
USAGE	A minimal registerable <code>Wsreg_component</code> configuration must include a version, unique name, display name, and an install location.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_add_display_name(3WSREG)</code> , <code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_register(3WSREG)</code> , <code>wsreg_set_id(3WSREG)</code> , <code>wsreg_set_location(3WSREG)</code> , <code>wsreg_set_unique_name(3WSREG)</code> , <code>wsreg_set_version(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_get, wsreg_get_all – query product install registry				
SYNOPSIS	<pre>cc [flag...] file ...-lwsreg [library...] #include <wsreg.h> Wsreg_component *wsreg_get(const Wsreg_query *query); Wsreg_component **wsreg_get_all(void);</pre>				
DESCRIPTION	The <code>wsreg_get()</code> function queries the product install registry for a component that matches the query specified by <i>query</i>. The <code>wsreg_get_all()</code> function returns all components currently registered in the product install registry.				
RETURN VALUES	The <code>wsreg_get()</code> function returns a pointer to a <code>Wsreg_component</code> structure representing the registered component. If no component matching the specified query is currently registered, <code>wsreg_get()</code> returns <code>NULL</code>. The <code>wsreg_get_all()</code> function returns a null-terminated array of <code>Wsreg_component</code> pointers. Each element in the resulting array represents one registered component.				
USAGE	The <code>wsreg</code> library must be initialized by a call to <code>wsreg_initialize(3WSREG)</code> before any call to <code>wsreg_get()</code> or <code>wsreg_get_all()</code>. The <code>Wsreg_component</code> pointer returned from <code>wsreg_get()</code> should be released through a call to <code>wsreg_free_component()</code>. See <code>wsreg_create_component(3WSREG)</code>. The <code>Wsreg_component</code> pointer array returned from <code>wsreg_get_all()</code> should be released through a call to <code>wsreg_free_component_array()</code>. See <code>wsreg_create_component(3WSREG)</code>.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_create_component(3WSREG)</code>, <code>wsreg_initialize(3WSREG)</code>, <code>wsreg_register(3WSREG)</code>, <code>attributes(5)</code>				

wsreg_initialize(3WSREG)

NAME	wsreg_initialize – initialize wsreg library				
SYNOPSIS	<pre>cc [flag...] file ...-lwsreg [library...] #include <wsreg.h> int wsreg_initialize(Wsreg_init_level <i>level</i>, const char *<i>alternate_root</i>) ;</pre>				
DESCRIPTION	The <code>wsreg_initialize()</code> function initializes the wsreg library. The <i>level</i> argument can be one of the following: <table><tr><td>WSREG_INIT_NORMAL</td><td>If an old registry file is present, attempt to perform a conversion.</td></tr><tr><td>WSREG_INIT_NO_CONVERSION</td><td>If an old conversion file is present, do not perform the conversion, but indicate that the conversion is recommended.</td></tr></table> The <i>alternate_root</i> argument can be used to specify a root prefix. If NULL is specified, no root prefix is used.	WSREG_INIT_NORMAL	If an old registry file is present, attempt to perform a conversion.	WSREG_INIT_NO_CONVERSION	If an old conversion file is present, do not perform the conversion, but indicate that the conversion is recommended.
WSREG_INIT_NORMAL	If an old registry file is present, attempt to perform a conversion.				
WSREG_INIT_NO_CONVERSION	If an old conversion file is present, do not perform the conversion, but indicate that the conversion is recommended.				
RETURN VALUES	The <code>wsreg_initialize()</code> function can return one of the following: <table><tr><td>WSREG_SUCCESS</td><td>The initialization was successful and no registry conversion is necessary.</td></tr><tr><td>WSREG_CONVERSION_RECOMMENDED</td><td>An old registry file exists and should be converted.</td></tr></table> A conversion is attempted if the <i>init_level</i> argument is WSREG_INIT_NORMAL and a registry file from a previous version of the product install registry exists. If the <code>wsreg_initialize()</code> function returns WSREG_CONVERSION_RECOMMENDED, the user either does not have permission to update the product install registry or does not have read/write access to the previous registry file.	WSREG_SUCCESS	The initialization was successful and no registry conversion is necessary.	WSREG_CONVERSION_RECOMMENDED	An old registry file exists and should be converted.
WSREG_SUCCESS	The initialization was successful and no registry conversion is necessary.				
WSREG_CONVERSION_RECOMMENDED	An old registry file exists and should be converted.				
USAGE	The <code>wsreg_initialize()</code> function must be called before any other wsreg library functions. The registry conversion can take some time to complete. The registry conversion can also be performed using the graphical registry viewer <code>/usr/bin/prodreg</code> or by the registry converter <code>/usr/bin/regconvert</code>.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Unsafe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>prodreg(1M)</code> , <code>wsreg_can_access_registry(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_query_create(3WSREG)

NAME	wsreg_query_create, wsreg_query_free – create a new query				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> Wsreg_query *wsreg_query_create(void); void wsreg_query_free(Wsreg_query *query);</pre>				
DESCRIPTION	The <code>wsreg_query_create()</code> function allocates a new query that can retrieve components from the product install registry. The <code>wsreg_query_free()</code> function releases the memory associated with the query specified by <i>query</i>.				
RETURN VALUES	The <code>wsreg_query_create()</code> function returns a pointer to the newly allocated query. The resulting query is completely empty and must be filled in to describe the desired component.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. The query must be configured and then passed to the <code>wsreg_get(3WSREG)</code> function to perform the registry query.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_query_set_id(3WSREG)</code> , <code>wsreg_query_set_instance(3WSREG)</code> , <code>wsreg_query_set_location(3WSREG)</code> , <code>wsreg_query_set_unique_name(3WSREG)</code> , <code>wsreg_query_set_version(3WSREG)</code> , <code>wsreg_unregister(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_query_set_id(3WSREG)

NAME	wsreg_query_set_id, wsreg_query_get_id – set or get the uuid of a query				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_query_set_id(Wsreg_query *query, const char *uuid); char *wsreg_query_get_id(const Wsreg_query *query);</pre>				
DESCRIPTION	The <code>wsreg_query_set_id()</code> function sets the uuid (universal unique identifier) specified by <i>uuid</i> in the query specified by <i>query</i>. If a uuid has already been set in the specified query, the resources associated with the previously set uuid are released. The <code>wsreg_query_get_id()</code> function returns the uuid associated with the query specified by <i>query</i>. The resulting string is not a copy and must not be released by the caller.				
RETURN VALUES	The <code>wsreg_query_set_id()</code> function returns non-zero if the uuid was set correctly; otherwise 0 is returned. The <code>wsreg_query_get_id()</code> function returns the uuid associated with the specified query.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. By specifying the uuid, the component search is narrowed to all components in the product install registry that have the specified uuid. Other fields can be specified in the same query to further narrow the search.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>towsreg_query_create(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_query_set_instance(3WSREG)

NAME	wsreg_query_set_instance, wsreg_query_get_instance – set or get the instance of a query				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_query_set_instance(Wsreg_query *query, int instance) ; int wsreg_query_get_instance(Wsreg_query *comp) ;</pre>				
DESCRIPTION	The <code>wsreg_query_set_instance()</code> function sets the instance number specified by <i>instance</i> in the query specified by <i>query</i>. The <code>wsreg_query_get_instance()</code> function retrieves the instance from the query specified by <i>query</i>.				
RETURN VALUES	The <code>wsreg_query_set_instance()</code> function returns a non-zero value if the instance was set correctly; otherwise 0 is returned. The <code>wsreg_query_get_instance()</code> function returns the instance number from the specified query. It returns 0 if the instance number has not been set.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. By specifying the instance, the component search is narrowed to all components in the product install registry that have the specified instance. Other fields can be specified in the same query to further narrow down the search.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_query_create(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_query_set_location(3WSREG)

NAME	wsreg_query_set_location, wsreg_query_get_location – set or get the location of a query				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_query_set_location(Wsreg_query *query, const char *location) ; char *wsreg_query_get_location(Wsreg_query *query) ;</pre>				
DESCRIPTION	The <code>wsreg_query_set_location()</code> function sets the location specified by <i>location</i> in the query specified by <i>query</i>. If a location has already been set in the specified query, the resources associated with the previously set location are released. The <code>wsreg_query_get_location()</code> function gets the location string from the query specified by <i>query</i>.				
RETURN VALUES	The <code>wsreg_query_set_location()</code> function returns a non-zero value if the location was set correctly; otherwise 0 is returned. The <code>wsreg_query_get_location()</code> function returns the location from the specified query structure. The resulting location string is not a copy, so it must not be released by the caller.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. By specifying the install location, the component search is narrowed to all components in the product install registry that are installed in the same location. Other fields can be specified in the same query to further narrow the search.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_query_create(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_query_set_unique_name, wsreg_query_get_unique_name – set or get the unique name of a query				
SYNOPSIS	<pre>cc [flag ...] file ...-lwsreg [library ...] #include <wsreg.h> int wsreg_query_set_unique_name(Wsreg_query *query, const char *unique_name) ; char *wsreg_query_get_unique_name(const Wsreg_query *query) ;</pre>				
DESCRIPTION	The <code>wsreg_query_set_unique_name()</code> function sets the unique name specified by <i>unique_name</i> in the query specified by <i>query</i>. If a unique name has already been set in the specified query, the resources associated with the previously set unique name are released. The <code>wsreg_query_get_unique_name()</code> function gets the unique name string from the query specified by <i>query</i>. The resulting string is not a copy and must not be released by the caller.				
RETURN VALUES	The <code>wsreg_query_set_unique_name()</code> function returns a non-zero value if the <i>unique_name</i> was set correctly; otherwise 0 is returned. The <code>wsreg_query_get_unique_name()</code> function returns a copy of the <i>unique_name</i> from the specified query.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. By specifying the unique name, the component search is narrowed to all components in the product install registry that have the specified unique name. Other fields can be specified in the same query to further narrow the search.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_query_create(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_query_set_version(3WSREG)

NAME	wsreg_query_set_version, wsreg_query_get_version – set or get the version of a query				
SYNOPSIS	<pre>cc [flag...] file... -lwsreg [library...] #include <wsreg.h> int wsreg_query_set_version(Wsreg_query *query, const char *version) ; char *wsreg_query_get_version(const Wsreg_query *query) ;</pre>				
DESCRIPTION	The <code>wsreg_query_set_version()</code> function sets the version specified by <i>version</i> in the query specified by <i>query</i>. If a version has already been set in the specified query, the resources associated with the previously set version are released. The <code>wsreg_query_get_version()</code> function gets the version string from the query specified by <i>query</i>. The resulting string is not a copy and must not be released by the caller.				
RETURN VALUES	The <code>wsreg_query_set_version()</code> function returns a non-zero value if the version was set correctly; otherwise 0 is returned. The <code>wsreg_query_get_version()</code> function returns the version from the specified query. If no version has been set, <code>NULL</code> is returned. The resulting version string is not a copy and must not be released by the caller.				
USAGE	The query identifies fields used to search for a specific component in the product install registry. By specifying the version, the component search is narrowed to all components in the product install registry that have the specified version. Other fields can be specified in the same query to further narrow the search.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_get(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_query_create(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_register – register a component in the product install registry
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_register(Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_register()</code> function updates a component in the product install registry. If <i>comp</i> is already in the product install registry, the call to <code>wsreg_register()</code> results in the currently registered component being updated. Otherwise, <i>comp</i> is added to the product install registry. An instance is assigned to the component upon registration. Subsequent component updates retain the same component instance. If <i>comp</i> has required components, each required component is updated to reflect the required component relationship. If <i>comp</i> has child components, each child component that does not already have a parent is updated to reflect specified component as its parent.
RETURN VALUES	Upon successful completion, a non-zero value is returned. If the component could not be updated in the product install registry, 0 is returned.
EXAMPLES	EXAMPLE 1 Create and register a component. The following example creates and registers a component. <pre>#include <wsreg.h> int main (int argc, char **argv) { char *uuid = "d6cf2869-1dd1-11b2-9fcb-080020b69971"; Wsreg_component *comp = NULL; /* Initialize the registry */ wsreg_initialize(WSREG_INIT_NORMAL, NULL); /* Create the component */ comp = wsreg_create_component(uuid); wsreg_set_unique_name(comp, "wsreg_example_1"); wsreg_set_version(comp, "1.0"); wsreg_add_display_name(comp, "en", "Example 1 component"); wsreg_set_type(comp, WSREG_COMPONENT); wsreg_set_location(comp, "/usr/local/example1_component"); /* Register the component */ wsreg_register(comp); wsreg_free_component(comp); return 0; }</pre>

wsreg_register(3WSREG)

USAGE A product's structure can be recorded in the product install registry by registering a component for each element and container in the product definition. The product and each of its features would be registered in the same way as a package that represents installed files.

Components should be registered only after they are successfully installed. If an entire product is being registered, the product should be registered after all components and features are installed and registered.

In order to register correctly, the component must be given a uuid, unique name, version, display name, and a location. The location assigned to product structure components should generally be the location in which the user chose to install the product.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO wsreg_get(3WSREG), wsreg_initialize(3WSREG),
wsreg_create_component(3WSREG), wsreg_unregister(3WSREG),
attributes(5)

NAME	wsreg_set_data, wsreg_get_data, wsreg_get_data_pairs – add or retrieve a key-value pair				
SYNOPSIS	<pre>cc [flag ...] file ... -lwsreg [library ...] #include <wsreg.h> int wsreg_set_data(Wsreg_component *comp, const char *key, const char *value) ; char *wsreg_get_data(const Wsreg_component *comp, const char *key) ; char *wsreg_get_data_pairs(const Wsreg_component *comp) ;</pre>				
DESCRIPTION	The <code>wsreg_set_data()</code> function adds the key-value pair specified by <i>key</i> and <i>value</i> to the component specified by <i>comp</i>. If <i>value</i> is <code>NULL</code>, the key and current value is removed from the specified component. The <code>wsreg_get_data()</code> function retrieves the value associated with the key specified by <i>key</i> from the component specified by <i>comp</i>. The <code>wsreg_get_data_pairs()</code> function returns the list of key-value pairs from the component specified by <i>comp</i>.				
RETURN VALUES	The <code>wsreg_set_data()</code> function returns a non-zero value if the specified key-value pair was successfully added. It returns 0 if the addition failed. If <code>NULL</code> is passed as the value, the current key-value pair are removed from the specified component. The <code>wsreg_get_data()</code> function returns the value associated with the specified key. It returns <code>NULL</code> if there is no value associated with the specified key. The char pointer that is returned is not a clone, so it must not be freed by the caller. The <code>wsreg_get_data_pairs()</code> function returns a null-terminated array of char pointers that represents the specified component's list of data pairs. The even indexes of the resulting array represent the key names. The odd indexes of the array represent the values. If the specified component has no data pairs, <code>NULL</code> is returned. The resulting array (not its contents) must be released by the caller.				
USAGE	Any string data can be associated with a component. Because this information can be viewed in the prodreg registry viewer, it is a good place to store support contact information. After the data pairs are added or removed, the component must be updated with a call to <code>wsreg_register(3WSREG)</code> for the modifications to be persistent.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				

`wsreg_set_data(3WSREG)`

SEE ALSO `prodreg(1M)`, `wsreg_initialize(3WSREG)`, `wsreg_register(3WSREG)`,
`attributes(5)`

NAME	wsreg_set_id, wsreg_get_id – set or get the uuid of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_id(Wsreg_component *comp, const char *uuid); char *wsreg_get_id(const Wsreg_component *comp);</pre>				
DESCRIPTION	The <code>wsreg_set_id()</code> function sets the uuid (universal unique identifier) specified by <i>uuid</i> into the component specified by <i>comp</i>. If a uuid has already been set into the specified component, the resources associated with the previously set uuid are released. The <code>wsreg_get_id()</code> function returns a copy of the uuid of the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_id()</code> function returns non-zero if the uuid was set correctly; otherwise 0 is returned. The <code>wsreg_get_id()</code> function returns a copy of the specified component's uuid.				
USAGE	Generally, the uuid will be set into a component by the <code>wsreg_create_component(3WSREG)</code> function, so a call to the <code>wsreg_set_id()</code> is not necessary.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:				
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_create_component(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_set_instance(3WSREG)

NAME	wsreg_set_instance, wsreg_get_instance – set or get the instance of a component
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_instance(Wsreg_component *comp, int instance); int wsreg_get_instance(Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_set_instance()</code> function sets the instance number specified by <i>instance</i> of the component specified by <i>comp</i>. The instance number and uuid are used to uniquely identify any component in the product install registry. The <code>wsreg_get_instance()</code> function determines the instance number associated with the component specified by <i>comp</i>.
RETURN VALUES	The <code>wsreg_set_instance()</code> function returns a non-zero value if the instance was set correctly; otherwise 0 is returned. The <code>wsreg_get_instance()</code> function returns the instance number associated with the specified component.
EXAMPLES	EXAMPLE 1 Get the instance value of a registered component. The following example demonstrates how how to get the instance value of a registered component. <pre>#include <fcntl.h> #include <wsreg.h> int main (int argc, char **argv) { char *uuid = "d6cf2869-1dd1-11b2-9fcb-080020b69971"; Wsreg_component *comp = NULL; /* Initialize the registry */ wsreg_initialize(WSREG_INIT_NORMAL, NULL); if (!wsreg_can_access_registry(O_RDWR)) { printf("No permission to modify the registry.\n"); return 1; } /* Create a component */ comp = wsreg_create_component(uuid); wsreg_set_unique_name(comp, "wsreg_example_1"); wsreg_set_version(comp, "1.0"); wsreg_add_display_name(comp, "en", "Example 1 component"); wsreg_set_type(comp, WSREG_COMPONENT); wsreg_set_location(comp, "/usr/local/example1_component"); /* Register */ wsreg_register(comp); printf("Instance %d was assigned\n", wsreg_get_instance(comp));</pre>

`wsreg_set_instance(3WSREG)`

EXAMPLE 1 Get the instance value of a registered component. *(Continued)*

```
wsreg_free_component(comp);  
return 0;  
}
```

USAGE Upon component registration with the `wsreg_register(3WSREG)` function, the instance number is set automatically. The instance number of 0 (the default) indicates to the `wsreg_register()` function that an instance number should be looked up and assigned during registration. If a component with the same uuid and location is already registered in the product install registry, that component's instance number will be used during registration.

After registration of a component, the `wsreg_get_instance()` function can be used to determine what instance value was assigned.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `wsreg_create_component(3WSREG)`, `wsreg_register(3WSREG)`, `attributes(5)`

wsreg_set_location(3WSREG)

NAME	wsreg_set_location, wsreg_get_location – set or get the location of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_location(Wsreg_component *comp, const char *location) ; char *wsreg_get_location(const Wsreg_component *comp) ;</pre>				
DESCRIPTION	The <code>wsreg_set_location()</code> function sets the location specified by <i>location</i> into the component specified by <i>comp</i>. Every component must have a location before being registered. If a location has already been set into the specified component, the resources associated with the previously set location are released. The <code>wsreg_get_location()</code> function gets the location string from the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_location()</code> function returns a non-zero value if the location was set correctly; otherwise 0 is returned. The <code>wsreg_get_location()</code> function returns a copy of the location from the specified component.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_set_parent, wsreg_get_parent – set or get the parent of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> void wsreg_set_parent(Wsreg_component *comp, const Wsreg_component *parent); Wsreg_component *wsreg_get_parent(const Wsreg_component *comp);</pre>				
DESCRIPTION	The <code>wsreg_set_parent()</code> function sets the parent specified by <i>parent</i> of the component specified by <i>comp</i>. The <code>wsreg_get_parent()</code> function gets the parent of the component specified by <i>comp</i>.				
RETURN VALUES	The <code>wsreg_get_parent()</code> function returns a pointer to a <code>Wsreg_component</code> structure that represents the parent of the specified component. If the specified component does not have a parent, NULL is returned. If a non-null value is returned, it the caller's responsibility to release the memory associated with the resulting <code>Wsreg_component</code> pointer with a call to <code>wsreg_free_component()</code> . See <code>wsreg_create_component(3WSREG)</code> .				
USAGE	The parent of a component is set as a result of registering the parent component. When a component that has children is registered, all of the child components are updated to reflect the newly registered component as their parent. This update only occurs if the child component does not already have a parent component set. The specified parent component is reduced to a lightweight component reference that uniquely identifies the parent in the product install registry. This lightweight reference includes the parent's uuid and instance number. The parent must be registered before a call to <code>wsreg_set_parent()</code> can be made, since the parent's instance number must be known at the time the <code>wsreg_set_parent()</code> function is called. A process needing to call <code>wsreg_set_parent()</code> or <code>wsreg_get_parent()</code> must have read access to the product install registry.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_can_access_registry(3WSREG)</code> , <code>wsreg_create_component(3WSREG)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>wsreg_register(3WSREG)</code> , <code>wsreg_set_instance(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_set_type(3WSREG)

NAME	wsreg_set_type, wsreg_get_type – set or get the type of a component						
SYNOPSIS	<pre>cc [<i>flag...</i>] <i>file...</i> -lwsreg [<i>library...</i>] #include <wsreg.h> int wsreg_set_type(Wsreg_component *<i>comp</i>, Wsreg_component_type <i>type</i>) ; Wsreg_component_type wsreg_get_type(const Wsreg_component *<i>comp</i>) ;</pre>						
DESCRIPTION	The <code>wsreg_set_type()</code> function sets the type specified by <i>type</i> in the component specified by <i>comp</i>. The <code>wsreg_get_type()</code> function retrieves the type from the component specified by <i>comp</i>.						
RETURN VALUES	The <code>wsreg_set_type()</code> function returns a non-zero value if the type is set successfully; otherwise 0 is returned. The <code>wsreg_get_type()</code> function returns the type currently set in the component specified by <i>comp</i>.						
USAGE	The component type is used to indicate whether a <code>Wsreg_component</code> structure represents a product, feature, or component. The <i>type</i> argument can be one of the following: <table> <tr> <td>WSREG_PRODUCT</td><td>Indicates the <code>Wsreg_component</code> represents a product. A product is a collection of features and/or components.</td></tr> <tr> <td>WSREG_FEATURE</td><td>Indicates the <code>Wsreg_component</code> represents a feature. A feature is a collection of components.</td></tr> <tr> <td>WSREG_COMPONENT</td><td>Indicates the <code>Wsreg_component</code> represents a component. A component is a collection of files that may be installed.</td></tr> </table>	WSREG_PRODUCT	Indicates the <code>Wsreg_component</code> represents a product. A product is a collection of features and/or components.	WSREG_FEATURE	Indicates the <code>Wsreg_component</code> represents a feature. A feature is a collection of components.	WSREG_COMPONENT	Indicates the <code>Wsreg_component</code> represents a component. A component is a collection of files that may be installed.
WSREG_PRODUCT	Indicates the <code>Wsreg_component</code> represents a product. A product is a collection of features and/or components.						
WSREG_FEATURE	Indicates the <code>Wsreg_component</code> represents a feature. A feature is a collection of components.						
WSREG_COMPONENT	Indicates the <code>Wsreg_component</code> represents a component. A component is a collection of files that may be installed.						
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Unsafe						
SEE ALSO	<code>wsreg_create_component(3WSREG)</code>, <code>wsreg_initialize(3WSREG)</code>, <code>wsreg_register(3WSREG)</code>, <code>wsreg_set_instance(3WSREG)</code>, <code>attributes(5)</code>						

NAME	wsreg_set_uninstaller, wsreg_get_uninstaller – set or get the uninstaller of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_uninstaller(Wsreg_component *comp, const char *uninstaller) ; char *wsreg_set_uninstaller(const Wsreg_component *comp) ;</pre>				
DESCRIPTION	The <code>wsreg_set_uninstaller()</code> function sets the uninstaller specified by <i>uninstaller</i> in the component specified by <i>comp</i>. If an uninstaller has already been set in the specified component, the resources associated with the previously set uninstaller are released. The <code>wsreg_get_uninstaller()</code> function gets the uninstaller string from the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_uninstaller()</code> function returns a non-zero value if the uninstaller was set correctly; otherwise 0 is returned. The <code>wsreg_get_uninstaller()</code> function returns a copy of the uninstaller from the specified component.				
USAGE	An uninstaller is usually only associated with a product, not with every component that comprises a product. The uninstaller string is a command that can be passed to the shell to launch the uninstaller. If an uninstaller is set in a registered component, the <code>prodreg(1M)</code> registry viewer will provide an uninstall button that will invoke the uninstaller.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>prodreg(1M)</code> , <code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_set_unique_name(3WSREG)

NAME	wsreg_set_unique_name, wsreg_get_unique_name – set or get the unique name of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_unique_name(Wsreg_component *comp, const char *unique_name) ; char *wsreg_get_unique_name(const Wsreg_component *comp) ;</pre>				
DESCRIPTION	The <code>wsreg_set_unique_name()</code> function sets the unique name specified by <i>unique_name</i> in the component specified by <i>comp</i>. Every component must have a unique name before being registered. If a unique name has already been set in the specified component, the resources associated with the previously set unique name are released. The <code>wsreg_get_unique_name()</code> function gets the unique name string from the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_unique_name()</code> function returns a non-zero value if the unique name was set correctly; otherwise it returns 0. The <code>wsreg_get_unique_name()</code> function returns a copy of the unique name from the specified component.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_set_vendor, wsreg_get_vendor – set or get the vendor of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_vendor(Wsreg_component *comp, const char *vendor); char *wsreg_get_vendor(const Wsreg_component *comp);</pre>				
DESCRIPTION	The <code>wsreg_set_vendor()</code> function sets the vendor specified by <i>vendor</i> in the component specified by <i>comp</i>. The <i>vendor</i> argument is a string that identifies the vendor of the component. If a vendor has already been set in the specified component, the resources associated with the previously set vendor are released. The <code>wsreg_get_vendor()</code> function gets the vendor string from the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_vendor()</code> function returns a non-zero value if the vendor was set correctly; otherwise it returns 0. The <code>wsreg_get_vendor()</code> function returns a copy of the vendor from the specified component.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

wsreg_set_version(3WSREG)

NAME	wsreg_set_version, wsreg_get_version – set or get the version of a component				
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_set_version(Wsreg_component *comp, const char *version); char *wsreg_get_version(const Wsreg_component *comp);</pre>				
DESCRIPTION	The <code>wsreg_set_version()</code> function sets the version specified by <i>version</i> in the component specified by <i>comp</i>. The <i>version</i> argument is a string that represents the version of the component. Every component must have a version before being registered. If a version has already been set in the specified component, the resources associated with the previously set version are released. The <code>wsreg_get_version()</code> function gets the version string from the component specified by <i>comp</i>. The resulting string must be released by the caller.				
RETURN VALUES	The <code>wsreg_set_version()</code> function returns a non-zero value if the version was set correctly; otherwise it returns 0. The <code>wsreg_get_version()</code> function returns a copy of the version from the specified component.				
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes: <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Unsafe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SEE ALSO	<code>wsreg_initialize(3WSREG)</code> , <code>attributes(5)</code>				

NAME	wsreg_unregister – remove a component from the product install registry
SYNOPSIS	<pre>cc [flag...] file...-lwsreg [library...] #include <wsreg.h> int wsreg_unregister(const Wsreg_component *comp);</pre>
DESCRIPTION	The <code>wsreg_unregister()</code> function removes the component specified by <i>comp</i> from the product install registry. The component will only be removed if the <i>comp</i> argument has a matching uuid, instance, and version. Usually, the component retrieved through a call to <code>wsreg_get(3WSREG)</code> before being passed to the <code>wsreg_unregister()</code> function. If the component has required components, the respective dependent components will be updated to reflect the change. A component that has dependent components cannot be unregistered until the dependent components are uninstalled and unregistered.
RETURN VALUES	Upon successful completion, a non-zero return value is returned. If the component could not be unregistered, 0 is returned.
EXAMPLES	EXAMPLE 1 Unregister a component. The following example demonstrates how to unregister a component. <pre>#include <stdio.h> #include <wsreg.h> int main(int argc, char **argv) { char *uuid = "d6cf2869-1dd1-11b2-9fcb-080020b69971"; char *location = "/usr/local/example1_component"; Wsreg_query *query = NULL; Wsreg_component *comp = NULL; /* Initialize the registry */ wsreg_initialize(WSREG_INIT_NORMAL, NULL); /* Query for the component */ query = wsreg_query_create(); wsreg_query_set_id(query, uuid); wsreg_query_set_location(query, location); comp = wsreg_get(query); if (comp != NULL) { /* The query succeeded. The component has been found. */ Wsreg_component **dependent_comps; dependent_comps = wsreg_get_dependent_components(comp); if (dependent_comps != NULL) { /* * The component has dependent components. The * component cannot be unregistered. */ wsreg_free_component_array(dependent_comps); } } }</pre>

wsreg_unregister(3WSREG)

EXAMPLE 1 Unregister a component. *(Continued)*

```
    printf("The component cannot be uninstalled because "
        "it has dependent components\n");
    } else {
        /*
         * The component does not have dependent components.
         * It can be unregistered.
         */
        if (wsreg_unregister(comp) != 0) {
            printf("wsreg_unregister succeeded\n");
        } else {
            printf("unregister failed\n");
        }
    }
    /* Be sure to free the component */
    wsreg_free_component(comp);
} else {
    /* The component is not currently registered. */
    printf("The component was not found in the registry\n");
}
wsreg_query_free(query);
}
```

USAGE Components should be unregistered before uninstallation. If the component cannot be unregistered, uninstallation should not be performed.

A component cannot be unregistered if other registered components require it. A call to `wsreg_get_dependent_components()` can be used to determine if this situation exists. See `wsreg_add_dependent_component(3WSREG)`.

A successful unregistration of a component will result in all components required by the unregistered component being updated in the product install registry to remove the dependency. Also, child components will be updated so the unregistered component is no longer registered as their parent.

When unregistering a product, the product should first be unregistered, followed by the unregistration of its first feature and then the unregistration and uninstallation of the components that comprise that feature. Be sure to use this top-down approach to avoid removing a component that belongs to a product or feature that is required by a separate product.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SEE ALSO `wsreg_add_dependent_component(3WSREG)`, `wsreg_get(3WSREG)`, `wsreg_initialize(3WSREG)`, `wsreg_register(3WSREG)`, `attributes(5)`

NAME	y0, y1, yn – Bessel functions of the second kind				
SYNOPSIS	<pre>cc [flag ...] file ... -lm [library ...]</pre> <pre>double y0(double x);</pre> <pre>double y1(double x);</pre> <pre>double yn(int n, double x);</pre>				
DESCRIPTION	The y0(), y1() and yn() functions compute Bessel functions of x of the second kind of orders 0, 1 and n respectively. The value of x must be positive.				
RETURN VALUES	Upon successful completion, y0(), y1() and yn() will return the relevant Bessel value of x of the second kind. If x is NaN, NaN is returned. If the x argument to y0(), y1() or yn() is negative, -HUGE_VAL or NaN is returned, and errno may be set to EDOM. If x is 0.0, -HUGE_VAL is returned and errno may be set to ERANGE or EDOM. If the correct result would cause overflow, -HUGE_VAL is returned and errno may be set to ERANGE. For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.				
ERRORS	The y0(), y1() and yn() functions may fail if: <table> <tr> <td>EDOM</td><td>The value of x is negative.</td></tr> <tr> <td>ERANGE</td><td>The value of x is too large in magnitude, or x is 0.0, or the correct result would cause overflow.</td></tr> </table>	EDOM	The value of x is negative.	ERANGE	The value of x is too large in magnitude, or x is 0.0, or the correct result would cause overflow.
EDOM	The value of x is negative.				
ERANGE	The value of x is too large in magnitude, or x is 0.0, or the correct result would cause overflow.				
USAGE	An application wishing to check for error situations should set errno to 0 before calling y0(), y1() or yn(). If errno is non-zero on return, or the return value is NaN, an error has occurred.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:				
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SEE ALSO	isnan(3M), j0(3M), matherr(3M), attributes(5), standards(5)				

y0(3M)

Index

A

- absolute value function — `fabs`, 204
- access a table property —
 - `picl_get_next_by_col`, 438
- access a table property —
 - `picl_get_next_by_row`, 438
- access CPU performance counters in other processes — `cpc_pctx_bind_event`, 79
- access CPU performance counters in other processes — `cpc_pctx_invalidate`, 79
- access CPU performance counters in other processes — `cpc_pctx_rele`, 79
- access CPU performance counters in other processes — `cpc_pctx_take_sample`, 79
- access PROM device information —
 - `di_prom_prop_data`, 104
- access PROM device information —
 - `di_prom_prop_name`, 104
- access PROM device information —
 - `di_prom_prop_next`, 104
- access property values and attributes —
 - `di_prop_bytes`, 108
- access property values and attributes —
 - `di_prop_devt`, 108
- access property values and attributes —
 - `di_prop_int64`, 108
- access property values and attributes —
 - `di_prop_ints`, 108
- access property values and attributes —
 - `di_prop_name`, 108
- access property values and attributes —
 - `di_prop_strings`, 108
- access property values and attributes —
 - `di_prop_type`, 108
- `aclcheck` — check the validity of an ACL, 22
- `aclfrommode` — convert an ACL to or from permission bits, 25
- `aclfromtext` — convert internal representation to or from external representation, 26
- `aclsort` — sort an ACL, 24
- `acltomode` — convert an ACL to or from permission bits, 25
- `acltotext` — convert internal representation to or from external representation, 26
- `acos` — arc cosine function, 28
- `acosh` — inverse hyperbolic functions, 29
- add or delete node to or from tree —
 - `ptree_add_node`, 490
- add or delete node to or from tree —
 - `ptree_delete_node`, 490
- add or remove a backward compatible version — `wsreg_add_compatible_version`, 682
- add or remove a backward compatible version — `wsreg_get_compatible_versions`, 682
- add or remove a backward compatible version — `wsreg_remove_compatible_version`, 682
- add or remove a child component —
 - `wsreg_add_child_component`, 680
- add or remove a child component —
 - `wsreg_get_child_components`, 680
- add or remove a child component —
 - `wsreg_remove_child_component`, 680
- add or remove a dependent component —
 - `wsreg_add_dependent_component`, 684

add or remove a dependent component —
 wsreg_get_dependent_components, 684
 add or remove a dependent component —
 wsreg_remove_dependent_component, 684
 add or remove a required component —
 wsreg_add_required_component, 688
 add or remove a required component —
 wsreg_get_required_components, 688
 add or remove a required component —
 wsreg_remove_required_component, 688
 add or retrieve a key-value pair —
 wsreg_get_data, 705
 add or retrieve a key-value pair —
 wsreg_get_data_pairs, 705
 add or retrieve a key-value pair —
 wsreg_set_data, 705
 add, remove, or return a localized display name
 — wsreg_add_display_name, 686
 add, remove, or return a localized display name
 — wsreg_get_display_languages, 686
 add, remove, or return a localized display name
 — wsreg_get_display_name, 686
 add, remove, or return a localized display name
 — wsreg_remove_display_name, 686
 advance — regular expression compile and
 match routines, 517
 allocate or deallocate a buffer for trace data
 — tnftcl_buffer_alloc, 625
 — tnftcl_buffer_dealloc, 625
 allow or disallow a memory segment to be
 imported by other nodes —
 rsm_memseg_export_publish, 535
 allow or disallow a memory segment to be
 imported by other nodes —
 rsm_memseg_export_republish, 535
 allow or disallow a memory segment to be
 imported by other nodes —
 rsm_memseg_export_unpublish, 535
 annotate source code with info for tools
 — NOTE, 367
 — _NOTE, 367
 arc cosine function — acos, 28
 arc sine function — asin, 30
 arc tangent function — atan2, 31
 arc tangent function — atan, 32
 asin — arc sine function, 30
 asinh — inverse hyperbolic functions, 29
 associate callbacks with process events —
 pctx_set_events, 430
 atan — arc tangent function, 32
 atan2 — arc tangent function, 31
 atanh — inverse hyperbolic functions, 29
 au_close — construct audit records, 33
 au_open — construct audit records, 33
 au_preselect — preselect an audit record, 35
 au_to — create audit record tokens, 37, 38
 au_to_arg — create audit record tokens, 37, 38
 au_to_arg32 — create audit record tokens, 37
 au_to_arg64 — create audit record tokens, 37
 au_to_attr — create audit record tokens, 37, 38
 au_to_cmd — create audit record tokens, 37
 au_to_data — create audit record tokens, 37, 38
 au_to_groups — create audit record tokens, 37,
 38
 au_to_in_addr — create audit record
 tokens, 37, 38
 au_to_ipc — create audit record tokens, 37, 38
 au_to_ipc_perm — create audit record
 tokens, 38
 au_to_iport — create audit record tokens, 37,
 38
 au_to_me — create audit record tokens, 37, 38
 au_to_new_in_addr — create audit record
 tokens, 38
 au_to_new_process — create audit record
 tokens, 38
 au_to_new_socket — create audit record
 tokens, 38
 au_to_new_subject — create audit record
 tokens, 38
 au_to_newgroups — create audit record
 tokens, 37
 au_to_opaque — create audit record tokens, 37,
 38
 au_to_path — create audit record tokens, 37, 38
 au_to_process — create audit record tokens, 37,
 38
 au_to_process_ex — create audit record
 tokens, 37
 au_to_return — create audit record tokens, 37,
 38
 au_to_return32 — create audit record
 tokens, 37
 au_to_return64 — create audit record
 tokens, 37

- au_to_socket — create audit record tokens, 37, 38
- au_to_subject — create audit record tokens, 37, 38
- au_to_subject_ex — create audit record tokens, 37
- au_to_text — create audit record tokens, 37, 38
- au_user_mask — get user's binary preselection mask, 41
- au_write — write audit records, 33
- audit control file information
 - endac, 213
 - getacdir, 213
 - getacflg, 213
 - getacinfo, 213
 - getacmin, 213
 - getacna, 213
 - setac, 213
- audit record tokens, manipulating
 - au_close, 33
 - au_open, 33
 - au_preselect, 35
 - au_write, 33
- authentication information routines for PAM
 - pam_get_item, 405
 - pam_set_item, 405
- authentication transaction routines for PAM
 - pam_end, 422
 - pam_start, 422

B

- base 10 logarithm function — log10, 323
- Basic Security Module functions
 - au_close, 33
 - au_open, 33
 - au_preselect, 35
 - au_user_mask, 41
 - au_write, 33
- Bessel functions of the first kind
 - j0, 269
 - j1, 269
 - jn, 269
- Bessel functions of the second kind
 - y0, 719
 - y1, 719
 - yn, 719

- bgets — read stream up to next delimiter, 43
- bind or unbind subscriber handle —
 - sysevent_bind_handle, 610
- bind or unbind subscriber handle —
 - sysevent_unbind_handle, 610
- buffer, split into fields — bufsplit, 45

C

- cbrt — cube root function, 46
- ceil — ceiling value function, 47
- ceiling value function — ceil, 47
- change or add a value to the PAM environment
 - pam_putenv, 399
- check for the presence of a smart card —
 - IFDHICCPresence, 258
- check the validity of an ACL — aclcheck, 22
- check whether or not Volume Management is managing a pathname — volmgt_inuse, 673
- check whether specific Volume Management features are enabled —
 - volmgt_feature_enabled, 672
- chkauthattr — verify user authorization, 220
- class-dependent data translation
 - elf32_xlatetof, 159
 - elf32_xlatetom, 159
 - elf64_xlatetof, 159
 - elf64_xlatetom, 159
- clone a component —
 - wsreg_clone_component, 692
- close a smartcard session, terminal, or card —
 - SCF_Session_close, 564
- close a smartcard session, terminal, or card —
 - SCF_Terminal_close, 564
- close a tnftcl handle — tnftcl_close, 627
- close the communication channel with an IFD
 - IFDHCloseChannel, 251
- commands, open, close to and from a command
 - p2open, p2close, 383
- compile — regular expression compile and match routines, 517
- compute natural logarithm — log1p, 324
- computes exponential functions — expm1, 203
- config_admin — configuration administration interface, 49
- config_ap_id_cmp — configuration administration interface, 49

config_change_state — configuration administration interface, 49
 config_list — configuration administration interface, 49
 config_list_ext — configuration administration interface, 49
 config_private_func — configuration administration interface, 49
 config_stat — configuration administration interface, 49
 config_strerror — configuration administration interface, 49
 config_test — configuration administration interface, 49
 config_unload_libs — configuration administration interface, 49
 configuration administration interface — config_admin, 49
 configuration administration interface — config_ap_id_cmp, 49
 configuration administration interface — config_change_state, 49
 configuration administration interface — config_list, 49
 configuration administration interface — config_list_ext, 49
 configuration administration interface — config_private_func, 49
 configuration administration interface — config_stat, 49
 configuration administration interface — config_strerror, 49
 configuration administration interface — config_test, 49
 configuration administration interface — config_unload_libs, 49
 connect to a DMI service provider — ConnectToServer, 56, 114
 construct, read, and write extended accounting records — ea_copy_object, 141
 construct, read, and write extended accounting records — ea_copy_object_tree, 141
 construct, read, and write extended accounting records — ea_get_creator, 141
 construct, read, and write extended accounting records — ea_get_hostname, 141
 construct, read, and write extended accounting records — ea_get_object, 141
 construct, read, and write extended accounting records — ea_get_object_tree, 141
 construct, read, and write extended accounting records — ea_next_object, 141
 construct, read, and write extended accounting records — ea_pack_object, 141
 construct, read, and write extended accounting records — ea_previous_object, 141
 construct, read, and write extended accounting records — ea_unpack_object, 141
 construct, read, and write extended accounting records — ea_write_object, 141
 control kernel tracing and process filtering — tnftcl_filter_list_add, 651
 — tnftcl_filter_list_delete, 651
 — tnftcl_filter_list_get, 651
 — tnftcl_filter_state_set, 651
 — tnftcl_trace_state_set, 651
 control probes of another process where caller provides /proc functionality — tnftcl_check_libs, 629
 — tnftcl_indirect_open, 629
 convert an ACL to or from permission bits — aclfrommode, 25
 convert an ACL to or from permission bits — acltmode, 25
 convert internal representation to or from external representation — aclfromtext, 26
 convert internal representation to or from external representation — acltotext, 26
 convert a supplied name into an absolute pathname that can be used to access removable media — media_findname, 340
 convert between Volume Management symbolic names, and the devices that correspond to them — volmgt_symdev, 678
 — volmgt_symname, 678
 coordinate CPC library and application versions — cpc_version, 87
 coordinate library and application versions — lgrp_version, 306
 copysign — return magnitude of first argument and sign of second argument, 58
 cos — cosine function, 59
 cosh — hyperbolic cosine function, 60
 cosine function — cos, 59
 cpc — hardware performance counters, 61

cpc_access — test access CPU performance counters, 64
 cpc_bind_event — use CPU performance counters on lwps, 65
 cpc_count_sys_events — enable and disable performance counters, 71
 cpc_count_usr_events — enable and disable performance counters, 71
 cpc_event — data structure to describe CPU performance counters, 73
 cpc_event_accum — simple difference and accumulate operations, 75
 cpc_event_diff — simple difference and accumulate operations, 75
 cpc_eventtostr — translate strings to and from events, 84
 cpc_getcciname — determine CPU performance counter configuration, 77
 cpc_getcpuref — determine CPU performance counter configuration, 77
 cpc_getcpuver — determine CPU performance counter configuration, 77
 cpc_getnpic — determine CPU performance counter configuration, 77
 cpc_getusage — determine CPU performance counter configuration, 77
 cpc_pctx_bind_event — access CPU performance counters in other processes, 79
 cpc_pctx_invalidate — access CPU performance counters in other processes, 79
 cpc_pctx_rele — access CPU performance counters in other processes, 79
 cpc_pctx_take_sample — access CPU performance counters in other processes, 79
 cpc_rele — use CPU performance counters on lwps, 65
 cpc_strtoevent — translate strings to and from events, 84
 cpc_take_sample — use CPU performance counters on lwps, 65
 cpc_version — coordinate CPC library and application versions, 87
 cpc_walk_names — determine CPU performance counter configuration, 77
 cplus_demangle — decode a C++ encoded symbol name, 88
 create a new query — wsreg_query_create, 697
 create a new query — wsreg_query_free, 697
 create audit record tokens — au_to, 37, 38
 create audit record tokens — au_to_arg32, 37
 create audit record tokens — au_to_arg64, 37
 create audit record tokens — au_to_arg, 37, 38
 create audit record tokens — au_to_attr, 37, 38
 create audit record tokens — au_to_cmd, 37
 create audit record tokens — au_to_data, 37, 38
 create audit record tokens — au_to_groups, 37, 38
 create audit record tokens — au_to_in_addr, 37, 38
 create audit record tokens — au_to_ipc, 37, 38
 create audit record tokens — au_to_ipc_perm, 38
 create audit record tokens — au_to_iport, 37, 38
 create audit record tokens — au_to_me, 37, 38
 create audit record tokens — au_to_newgroups, 37
 create audit record tokens — au_to_new_in_addr, 38
 create audit record tokens — au_to_new_process, 38
 create audit record tokens — au_to_new_socket, 38
 create audit record tokens — au_to_new_subject, 38
 create audit record tokens — au_to_opaque, 37, 38
 create audit record tokens — au_to_path, 37, 38
 create audit record tokens — au_to_process, 37, 38
 create audit record tokens — au_to_process_ex, 37
 create audit record tokens — au_to_return32, 37
 create audit record tokens — au_to_return64, 37
 create audit record tokens — au_to_return, 37, 38
 create audit record tokens — au_to_socket, 37, 38
 create audit record tokens — au_to_subject, 37, 38
 create audit record tokens — au_to_subject_ex, 37
 create audit record tokens — au_to_text, 37, 38

- create or break logical connection between import and export segments — `rsm_memseg_import_connect`, 539
- create or break logical connection between import and export segments — `rsm_memseg_import_disconnect`, 539
- create or destroy a property — `ptree_create_prop`, 496
- create or destroy a property — `ptree_destroy_prop`, 496
- create or destroy barrier for imported segment — `rsm_memseg_import_destroy_barrier`, 543
- create or destroy barrier for imported segment — `rsm_memseg_import_init_barrier`, 543
- create or free local memory handle — `rsm_create_localmemory_handle`, 522
- create or free local memory handle — `rsm_free_localmemory_handle`, 522
- create or release a component — `wsreg_create_component`, 694
- create or release a component — `wsreg_free_component`, 694
- create or release a component — `wsreg_free_component_array`, 694
- create, destroy and manipulate exact objects — `ea_alloc`, 146
- create, destroy and manipulate exact objects — `ea_attach_to_group`, 146
- create, destroy and manipulate exact objects — `ea_attach_to_object`, 146
- create, destroy and manipulate exact objects — `ea_free`, 146
- create, destroy and manipulate exact objects — `ea_free_item`, 146
- create, destroy and manipulate exact objects — `ea_free_object`, 146
- create, destroy and manipulate exact objects — `ea_match_object_catalog`, 146
- create, destroy and manipulate exact objects — `ea_set_group`, 146
- create, destroy and manipulate exact objects — `ea_set_item`, 146
- create, destroy and manipulate exact objects — `ea_strdup`, 146
- create, destroy and manipulate exact objects — `ea_strfree`, 146

- create a communication channel with an IFD — `IFDHCreateChannel`, 253
- create a communication channel with an IFD — `IFDHCreateChannelByName`, 254
- create and add node to tree and return node handle — `ptree_create_and_add_node`, 492
- create and add property to node and return property handle — `ptree_create_and_add_prop`, 493
- create `DmiOctetString` in dynamic memory, — `newDmiOctetString`, 363
- create `DmiString` in dynamic memory, — `newDmiString`, 364
- create handle for internal process probe control — `tnfctl_internal_open`, 632
- create handle for kernel probe control — `tnfctl_kernel_open`, 634
- cube root function — `cbrt`, 46

D

- data structure to describe CPU performance counters — `cpc_event`, 73
- decode a C++ encoded symbol name — `cplus_demangle`, 88
- `demangle`, 88
- demangle — decode a C++ encoded symbol name, 88
- destroy a layout object — `m_destroy_layout`, 339
- determine CPU performance counter configuration — `cpc_getcciname`, 77
- determine CPU performance counter configuration — `cpc_getcpuref`, 77
- determine CPU performance counter configuration — `cpc_getcpuver`, 77
- determine CPU performance counter configuration — `cpc_getnpic`, 77
- determine CPU performance counter configuration — `cpc_getusage`, 77
- determine CPU performance counter configuration — `cpc_walk_names`, 77
- determine access to product install registry — `wsreg_can_access_registry`, 690
- determine equality of two components — `wsreg_components_equal`, 693

determine whether snapshot of lgroup hierarchy is stale — `lgrp_cookie_stale`, 296

device ID interfaces for user applications — `devid_compare`, 89

device ID interfaces for user applications — `devid_deviceid_to_nmlist`, 89

device ID interfaces for user applications — `devid_free`, 89

device ID interfaces for user applications — `devid_free_nmlist`, 89

device ID interfaces for user applications — `devid_get`, 89

device ID interfaces for user applications — `devid_get_minor_name`, 89

device ID interfaces for user applications — `devid_sizeof`, 89

device ID interfaces for user applications — `devid_str_decode`, 89

device ID interfaces for user applications — `devid_str_encode`, 89

device ID interfaces for user applications — `devid_str_free`, 89

device ID interfaces for user applications — `devid_valid`, 89

`devid_compare` — device ID interfaces for user applications, 89

`devid_deviceid_to_nmlist` — device ID interfaces for user applications, 89

`devid_free` — device ID interfaces for user applications, 89

`devid_free_nmlist` — device ID interfaces for user applications, 89

`devid_get` — device ID interfaces for user applications, 89

`devid_get_minor_name` — device ID interfaces for user applications, 89

`devid_sizeof` — device ID interfaces for user applications, 89

`devid_str_decode` — device ID interfaces for user applications, 89

`devid_str_encode` — device ID interfaces for user applications, 89

`devid_str_free` — device ID interfaces for user applications, 89

`devid_valid` — device ID interfaces for user applications, 89

`di_minor_devt` — return libdevinfo minor node information, 101

`di_minor_name` — return libdevinfo minor node information, 101

`di_minor_nodetype` — return libdevinfo minor node information, 101

`di_minor_spectype` — return libdevinfo minor node information, 101

`di_prom_prop_data` — access PROM device information, 104

`di_prom_prop_name` — access PROM device information, 104

`di_prom_prop_next` — access PROM device information, 104

`di_prop_bytes` — access property values and attributes, 108

`di_prop_devt` — access property values and attributes, 108

`di_prop_int64` — access property values and attributes, 108

`di_prop_ints` — access property values and attributes, 108

`di_prop_lookup_bytes` — search for a property, 111

`di_prop_lookup_int64` — search for a property, 111

`di_prop_lookup_ints` — search for a property, 111

`di_prop_lookup_strings` — search for a property, 111

`di_prop_name` — access property values and attributes, 108

`di_prop_strings` — access property values and attributes, 108

`di_prop_type` — access property values and attributes, 108

`di_walk_minor` — traverse libdevinfo minor nodes, 115

`di_walk_node` — traverse libdevinfo device nodes, 117

directories, create, remove them in a path — `mkdirp`, `rmdirp`, 347

`dmi_error` — print error in string form, 127

`DmiAddComponent` — Management Interface database administration functions, 118

`DmiAddGroup` — Management Interface database administration functions, 118

`DmiAddLanguage` — Management Interface database administration functions, 118

DmiAddRow — Management Interface
 operation functions, 122
 DmiDeleteComponent — Management
 Interface database administration
 functions, 118
 DmiDeleteGroup — Management Interface
 database administration functions, 118
 DmiDeleteLanguage — Management Interface
 database administration functions, 118
 DmiDeleteRow — Management Interface
 operation functions, 122
 DmiGetAttribute — Management Interface
 operation functions, 122
 DmiGetConfig — Management Interface
 initialization functions, 128
 DmiGetMultiple — Management Interface
 operation functions, 122
 DmiGetVersion — Management Interface
 initialization functions, 128
 DmiListAttributes — Management Interface
 listing functions, 131
 DmiListClassNames — Management Interface
 listing functions, 131
 DmiListComponents — Management Interface
 listing functions, 131
 DmiListComponentsByClass — Management
 Interface listing functions, 131
 DmiListGroup — Management Interface listing
 functions, 131
 DmiListLanguages — Management Interface
 listing functions, 131
 DmiOriginateEvent — Service Provider
 functions for components, 136
 DmiRegister — Management Interface
 initialization functions, 128
 DmiRegisterCi — Service Provider functions for
 components, 136
 DmiSetAttribute — Management Interface
 operation functions, 122
 DmiSetConfig — Management Interface
 initialization functions, 128
 DmiSetMultiple — Management Interface
 operation functions, 122
 DmiUnregister — Management Interface
 initialization functions, 128
 DmiUnregisterCi — Service Provider functions
 for components, 136

E

ea_alloc — create, destroy and manipulate
 exact objects, 146
 ea_attach_to_group — create, destroy and
 manipulate exact objects, 146
 ea_attach_to_object — create, destroy and
 manipulate exact objects, 146
 ea_close — open or close exact files, 139
 ea_copy_object — construct, read, and write
 extended accounting records, 141
 ea_copy_object_tree — construct, read, and
 write extended accounting records, 141
 ea_error — error interface to extended
 accounting library, 138
 ea_free — create, destroy and manipulate exact
 objects, 146
 ea_free_item — create, destroy and manipulate
 exact objects, 146
 ea_free_object — create, destroy and
 manipulate exact objects, 146
 ea_get_creator — construct, read, and write
 extended accounting records, 141
 ea_get_hostname — construct, read, and write
 extended accounting records, 141
 ea_get_object — construct, read, and write
 extended accounting records, 141
 ea_get_object_tree — construct, read, and write
 extended accounting records, 141
 ea_match_object_catalog — create, destroy and
 manipulate exact objects, 146
 ea_next_object — construct, read, and write
 extended accounting records, 141
 ea_open — open or close exact files, 139
 ea_pack_object — construct, read, and write
 extended accounting records, 141
 ea_previous_object — construct, read, and write
 extended accounting records, 141
 ea_set_group — create, destroy and manipulate
 exact objects, 146
 ea_set_item — create, destroy and manipulate
 exact objects, 146
 ea_strdup — create, destroy and manipulate
 exact objects, 146
 ea_strfree — create, destroy and manipulate
 exact objects, 146
 ea_unpack_object — construct, read, and write
 extended accounting records, 141

ea_write_object — construct, read, and write extended accounting records, 141
 elf — object file access library, 161
 elf, get entries from name list — nlist, 366
 elf_begin — process ELF object files, 167
 elf_cntl — control an elf file descriptor, 172
 elf_end — process ELF object files, 167
 elf_errmsg — error handling, 174
 elf_errno — error handling, 174
 elf_fill — set fill byte, 175
 elf_flagdata — manipulate flags, 176
 elf_flagehdr — manipulate flags, 176
 elf_flagelf — manipulate flags, 176
 elf_flagphdr — manipulate flags, 176
 elf_flagshdr — manipulate flags, 176
 elf_getarhdr — retrieve archive member header, 178
 elf_getarsym — retrieve archive symbol table, 180
 elf_getbase — get the base offset for an object file, 181
 elf_getdata — get section data, 182
 elf_getident — retrieve file identification data, 187
 elf_getscn — get section information, 189
 elf_hash — compute hash value, 191
 elf_kind — determine file type, 192
 elf_memory — process ELF object files, 167
 elf_ndxscn — get section information, 189
 elf_newdata — get section data, 182
 elf_newscn — get section information, 189
 elf_next — process ELF object files, 167
 elf_nextscn — get section information, 189
 elf_rand — process ELF object files, 167
 elf_rawdata — get section data, 182
 elf_rawfile — retrieve uninterpreted file contents, 193
 elf_strptr — make a string pointer, 195
 elf_update — update an ELF descriptor, 196
 elf_version — coordinate ELF library and application versions, 200
 elf32_checksum — return the checksum of an elf image, — elf64_checksum, 151
 elf32_fsize — return the size of an object file type, 152
 elf32_getehdr — retrieve class-dependent object file header, 153
 elf32_getphdr — retrieve class-dependent program header table, 155
 elf32_getshdr — retrieve class-dependent section header, 157
 elf32_newehdr — retrieve class-dependent object file header, 153
 elf32_newphdr — retrieve class-dependent program header table, 155
 elf32_xlatetof — class-dependent data translation, 159
 elf32_xlatetom — class-dependent data translation, 159
 elf64_checksum — return the checksum of an elf image, — elf32_checksum, 151
 elf64_fsize — return the size of an object file type, 152
 elf64_getehdr — retrieve class-dependent object file header, 153
 elf64_getphdr — retrieve class-dependent program header table, 155
 elf64_getshdr — retrieve class-dependent section header, 157
 elf64_newehdr — retrieve class-dependent object file header, 153
 elf64_newphdr — retrieve class-dependent program header table, 155
 elf64_xlatetof — class-dependent data translation, 159
 elf64_xlatetom — class-dependent data translation, 159
 enable and disable performance counters — cpc_count_sys_events, 71
 enable and disable performance counters — cpc_count_usr_events, 71
 encryption, determine whether a buffer of characters is encrypted — isencrypt, 267
 endac — get audit control file information, 213
 endaclass — close audit_class database file, 215
 endaevent — close audit_event database file, 218
 endauthattr — get authorization database entry, 220
 endauuser — get audit_user database entry, 223
 endddent — get device_deallocate entry, 225
 enddmapent — get device_maps entry, 227
 endexecattr — get execution profile entry, 229

- endprofattr — get profile description and attributes, 233
- endproject — project database entry functions, 235
- enduserattr — get user_attr entry, 239
- erf — error and complementary error functions, 201
- erfc — error and complementary error functions, 201
- error interface to resource pools library — pool_error, 465
- error interface to resource pools library — pool_strerror, 465
- error and complementary error functions — erf, 201
- erfc, 201
- error interface to extended accounting library — ea_error, 138
- establish a context with a smartcard — SCF_Terminal_getCard, 584
- establish a context with a smartcard terminal (reader) — SCF_Session_getTerminal, 574
- establish a context with a system's smartcard framework — SCF_Session_getSession, 572
- Euclidean distance function — hypot, 242
- Executable and Linking Format, *See* elf
- exp — exponential function, 202
- expm1 — computes exponential functions, 203
- exponential function — exp, 202

F

- fabs — absolute value function, 204
- fgetproject — project database entry functions, 235
- files, search for named file in named directories — pathfind, 426
- find node with given property and value — ptree_find_node, 499
- finished using lgroup interface — lgrp_fini, 298
- floating-point remainder value function — fmod, 206
- floor — floor function, 205
- floor function — floor, 205
- fmod — floating-point remainder value function, 206
- free_authattr — release memory, 220

- free dynamic memory allocated for input DmiString structure, — freeDmiString, 207
- free_execattr — get execution profile entry, 229
- free memory for sysevent handle — sysevent_free, 612
- free_profattr — get profile description and attributes, 233
- free_proflist — get execution profile entry, 229
- free_proflist — get profile description and attributes, 233
- free_userattr — get user_attr entry, 239
- freeDmiString — free dynamic memory allocated for input DmiString structure, 207
- functions to manage lockfile(s) for user's mailbox — maillock, 327
- mailunlock, 327
- touchlock, 327

G

- gamma — log gamma function, 291
- gamma_r — log gamma function, 291
- get a property handle of a node — picl_get_first_prop, 436
- get a property handle of a node — picl_get_next_prop, 436
- get class name, subclass name, ID or buffer size of event — sysevent_get_class_name, 614
- get class name, subclass name, ID or buffer size of event — sysevent_get_event_id, 614
- get class name, subclass name, ID or buffer size of event — sysevent_get_size, 614
- get class name, subclass name, ID or buffer size of event — sysevent_get_subclass_name, 614
- get device_deallocate entry — endddent, 225
- get device_deallocate entry — getddent, 225
- get device_deallocate entry — getddnam, 225
- get device_deallocate entry — setddent, 225
- get device_deallocate entry — setddfile, 225
- get device_maps entry — enddmapent, 227
- get device_maps entry — getdmapent, 227
- get device_maps entry — getdmapnam, 227
- get device_maps entry — getdmaptddev, 227
- get device_maps entry — getdmaptype, 227
- get device_maps entry — setdmapent, 227

get device_maps entry — setdmapfile, 227
 get execution profile entry — endexecattr, 229
 get execution profile entry — free_execattr, 229
 get execution profile entry — free_proflist, 229
 get execution profile entry — getexecattr, 229
 get execution profile entry — getexecprof, 229
 get execution profile entry — getexecuser, 229
 get execution profile entry — get_profiles, 229
 get execution profile entry —
 match_execattr, 229
 get execution profile entry — setexecattr, 229
 get of set lgroup affinity —
 lgrp_affinity_get, 293
 get of set lgroup affinity —
 lgrp_affinity_set, 293
 get or free interconnect topology —
 rsm_free_interconnect_topology, 526
 get or free interconnect topology —
 rsm_get_interconnect_topology, 526
 get or release a controller handle —
 rsm_get_controller, 524
 get or release a controller handle —
 rsm_get_controller_attr, 524
 get or release a controller handle —
 rsm_release_controller, 524
 get or release a poll descriptor —
 rsm_memseg_get_pollfd, 538
 get or release a poll descriptor —
 rsm_memseg_release_pollfd, 538
 get profile description and attributes —
 endprofattr, 233
 get profile description and attributes —
 free_profattr, 233
 get profile description and attributes —
 free_proflist, 233
 get profile description and attributes —
 getprofattr, 233
 get profile description and attributes —
 getproflist, 233
 get profile description and attributes —
 getprofnam, 233
 get profile description and attributes —
 setprofattr, 233
 get section data — elf_getdata, 182
 get section data — elf_newdata, 182
 get section data — elf_rawdata, 182
 get the value of a property —
 picl_get_propval, 442
 get the value of a property —
 picl_get_propval_by_name, 442
 get the value of a property —
 ptree_get_propval, 507
 get the value of a property —
 ptree_get_propval_by_name, 507
 get user_attr entry — enduserattr, 239
 get user_attr entry — free_userattr, 239
 get user_attr entry — getuserattr, 239
 get user_attr entry — getusernam, 239
 get user_attr entry — getuserid, 239
 get user_attr entry — setuserattr, 239
 get vendor name, publisher name or processor
 ID of event — sysevent_get_pid, 616
 get vendor name, publisher name or processor
 ID of event — sysevent_get_pub_name, 616
 get vendor name, publisher name or processor
 ID of event —
 sysevent_get_vendor_name, 616
 get a string describing a status code —
 SCF_strerror, 577
 get and set media attributes
 — media_getattr, 343
 — media_setattr, 343
 get attribute list pointer —
 sysevent_get_attr_list, 613
 get children of given lgroup —
 lgrp_children, 295
 get CPU IDs contained in specified lgroup —
 lgrp_cpus, 297
 get error message string — picl_strerror, 449
 get handle of node specified by PICL tree path
 — ptree_get_node_by_path, 502
 get home lgroup — lgrp_home, 299
 get IFD capabilities —
 IFDHGetCapabilities, 256
 get latency between two lgroups —
 lgrp_latency, 301
 get number of lgroups — lgrp_nlgtps, 303
 get parents of given lgroup —
 lgrp_parents, 304
 get_profiles — get execution profile entry, 229
 get property information —
 ptree_get_propinfo, 505
 get property information and handle of named
 property — picl_get_propinfo_by_name, 441

get property information and handle of named property —
 ptree_get_propinfo_by_name, 506
 get segment ID range —
 rsm_get_segmentid_range, 528
 get the handle of the property by name —
 picl_get_prop_by_name, 439
 get the information about a property —
 picl_get_propinfo, 440
 get the root handle of the PICL tree —
 picl_get_root, 444
 get the root node handle — ptree_get_root, 508
 get the trace attributes from a tnftcl handle —
 tnftcl_trace_attrs_get, 649
 get view of lgroup hierarchy — lgrp_view, 307
 getacdir — get audit control file information, 213
 getacflg — get audit control file information, 213
 getacinfo — get audit control file information, 213
 getacmin — get audit control file information, 213
 getacna — get audit control file information, 213
 getaclassent — get audit_class database entry, 215
 getaclassent_r — get audit_class database entry, 215
 getaclassnam — get audit_class database entry, 215
 getaclassnam_r — get audit_class database entry, 215
 getauditflags() — generate process audit state, 232
 getauditflagsbin() — convert audit flag specifications, 217
 getauditflagschar() — convert audit flag specifications, 217
 getauevent — get audit_event database entry, 218
 getauevent_r — get audit_event database entry, 218
 getauevnam — get audit_event database entry, 218
 getauevnam_r — get audit_event database entry, 218
 getauevnonam — get audit_event database entry, 218
 getauevnum — get audit_event database entry, 218
 getauevnum_r — get audit_event database entry, 218
 getauthattr — get authorization database entry, 220
 getauthnam — get authorization database entry, 220
 getauuserent — get audit_user database entry, 223
 getauuserent_r — get audit_user database entry, 223
 getauusernam — get audit_user database entry, 223
 getauusernam_r — get audit_user database entry, 223
 getddent — get device_deallocate entry, 225
 getddnam — get device_deallocate entry, 225
 getdefaultproj — project database entry functions, 235
 getdmapent — get device_maps entry, 227
 getdmapnam — get device_maps entry, 227
 getdmaptddev — get device_maps entry, 227
 getdmaptype — get device_maps entry, 227
 getexecattr — get execution profile entry, 229
 getexecprof — get execution profile entry, 229
 getexecuser — get execution profile entry, 229
 getprofattr — get profile description and attributes, 233
 getproflist — get profile description and attributes, 233
 getprofnam — get profile description and attributes, 233
 getprojbyid — project database entry functions, 235
 getprojbyname — project database entry functions, 235
 getproject — project database entry functions, 235
 getuserattr — get user_attr entry, 239
 getusernam — get user_attr entry, 239
 getuserid — get user_attr entry, 239
 gmatch — shell global pattern matching, 241

H

hardware performance counters — cpc, 61
have Volume Management check for media —
 volmgt_check, 670
hyperbolic cosine function — cosh, 60
hyperbolic sine function — sinh, 599
hyperbolic tangent function — tanh, 624
hypot — Euclidean distance function, 242

I

IFDHCloseChannel — close the communication
 channel with an IFD, 251
IFDHControl — send control information to an
 IFD, 252
IFDHCreateChannel — create a communication
 channel with an IFD, 253
IFDHCreateChannelByName — create a
 communication channel with an IFD, 254
IFDHGetCapabilities — get IFD
 capabilities, 256
IFDHICCPresence — check for the presence of a
 smart card, 258
IFDHPowerICC — power up or power down
 the smart card, 259
IFDHSetCapabilities — set slot or card
 capabilities, 261
IFDHSetProtocolParameters — set protocol
 parameters, 262
IFDHTransmitToICC — transmit APDU to a
 smart card, 264
ilogb — returns an unbiased exponent, 266
initialize a layout object —
 m_create_layout, 335
initialize kernel statistics facility
 — kstat_close, 278
 — kstat_open, 278
initialize lgroup interface — lgrp_init, 300
initialize ptree_propinfo_t structure —
 ptree_init_propinfo, 509
initialize wsreg library — wsreg_initialize, 696
initiate a session with the PICL daemon —
 picl_initialize, 445
inproj — project database entry functions, 235
interfaces for direct probe and process control
 for another process
 — tnftctl_continue, 635

interfaces for direct probe and process control
for another process (Continued)
 — tnftctl_exec_open, 635
 — tnftctl_pid_open, 635
interfaces to query and to change the state of a
 probe
 — tnftctl_probe_connect, 643
 — tnftctl_probe_disable, 643
 — tnftctl_probe_disconnect_all, 643
 — tnftctl_probe_enable, 643
 — tnftctl_probe_state_get, 643
 — tnftctl_probe_trace, 643
 — tnftctl_probe_untrace, 643
inverse hyperbolic functions
 — acosh, 29
 — asinh, 29
 — atanh, 29
isencrypt — determine whether a buffer of
 characters is encrypted, 267
isnan — test for NaN, 268
iterate over probes
 — tnftctl_probe_apply, 640
 — tnftctl_probe_apply_ids, 640

J

j0 — Bessel functions of the first kind, 269
j1 — Bessel functions of the first kind, 269
jn — Bessel functions of the first kind, 269

K

kernel virtual memory functions
 copy data from kernel image or running
 system — kvm_read, kvm_kread,
 kvm_uread, 288
 copy data to kernel image or running system
 — kvm_write, kvm_kwrite,
 kvm_uwrite, 288
 get invocation argument for process —
 kvm_getcmd, 281
 get u-area for process — kvm_getu, 281
 get entries from kernel symbol table —
 kvm_nlist, 285
 kstat — kernel statistics facility, 270

kernel virtual memory functions (Continued)

kstat_chain_update — update the kstat header chain, 276
kstat_close — initialize kernel statistics facility, 278
kstat_data_lookup — find a kstat by name, 277
kstat_lookup — find a kstat by name, 277
kstat_open — initialize kernel statistics facility, 278
kstat_read — read or write kstat data, 279
kstat_write — read or write kstat data, 279
specify a kernel to examine — kvm_open, kvm_close, 286
kstat — kernel statistics facility, 270
kstat_chain_update — update the kstat header chain, 276
kstat_close — initialize kernel statistics facility, 278
kstat_data_lookup — find a kstat by name, 277
kstat_lookup — find a kstat by name, 277
kstat_open — initialize kernel statistics facility, 278
kstat_read — read or write kstat data, 279
kstat_write — read or write kstat data, 279
kva_match — look up a key in a key-value array, 280
kvm_close — specify kernel to examine, 286
kvm_getcmd — get invocation arguments for process, 281
kvm_getproc — read system process structures, 283
kvm_getu — get u-area for process, 281
kvm_kread — copy data from a kernel image or running system, 288
kvm_kwrite — copy data to a kernel image or running system, 288
kvm_nextproc — read system process structures, 283
kvm_nlist — get entries from kernel symbol table, 285
kvm_open — specify kernel to examine, 286
kvm_read — copy data from kernel image or running system, 288
kvm_setproc — read system process structures, 283
kvm_uread — copy data from a kernel image or running system, 288

kvm_uwrite — copy data to a kernel image or running system, 288
kvm_write — copy data to kernel image or running system, 288

L

la_activity — runtime linker auditing functions, 553
la_i86_pltenter — runtime linker auditing functions, 553
la_objopen — runtime linker auditing functions, 553
la_objsearch — runtime linker auditing functions, 553
la_pltexit — runtime linker auditing functions, 553
la_pltexit64 — runtime linker auditing functions, 553
la_preinit — runtime linker auditing functions, 553
la_sparcv8_pltenter — runtime linker auditing functions, 553
la_sparcv9_pltenter — runtime linker auditing functions, 553
la_symbind32 — runtime linker auditing functions, 553
la_symbind64 — runtime linker auditing functions, 553
la_version — runtime linker auditing functions, 553
layout transformation —
 m_transform_layout, 352
layout transformation for wide character strings —
 m_wtransform_layout, 357
ld_atexit — link-editor support functions, 290
ld_atexit64 — link-editor support functions, 290
ld_file — link-editor support functions, 290
ld_file64 — link-editor support functions, 290
ld_input_done — link-editor support functions, 290
ld_input_section — link-editor support functions, 290
ld_input_section64 — link-editor support functions, 290
ld_section — link-editor support functions, 290

ld_section64 — link-editor support functions, 290
 ld_start — link-editor support functions, 290
 ld_start64 — link-editor support functions, 290
 ld_support — link-editor support functions, 290
 ld_version — link-editor support functions, 290
 lgamma — log gamma function, 291
 lgamma_r — log gamma function, 291
 lgrp_affinity_get — get of set lgroup affinity, 293
 lgrp_affinity_set — get of set lgroup affinity, 293
 lgrp_children — get children of given lgroup, 295
 lgrp_cookie_stale — determine whether snapshot of lgroup hierarchy is stale, 296
 lgrp_cpus — get CPU IDs contained in specified lgroup, 297
 lgrp_fini — finished using lgroup interface, 298
 lgrp_home — get home lgroup, 299
 lgrp_init — initialize lgroup interface, 300
 lgrp_latency — get latency between two lgroups, 301
 lgrp_nlgrps — get number of lgroups, 303
 lgrp_parents — get parents of given lgroup, 304
 lgrp_root — return root lgroup ID, 305
 lgrp_version — coordinate library and application versions, 306
 lgrp_view — get view of lgroup hierarchy, 307
 libdevinfo — library of device information functions, 308
 libnvpair — library of name—value pair functions, 311
 libpicl — PICL interface library, 312
 libpicltree — PTree and Plug-in Registration interface library, 315
 library for TNF probe control in a process or the kernel — libtnfctl, 318
 library of device information functions — libdevinfo, 308
 library of name—value pair functions — libnvpair, 311
 libtnfctl — library for TNF probe control in a process or the kernel, 318
 link-editor support functions —
 ld_atexit64, 290
 link-editor support functions — ld_atexit, 290
 link-editor support functions — ld_file64, 290
 link-editor support functions — ld_file, 290
 link-editor support functions —
 ld_input_done, 290
 link-editor support functions —
 ld_input_section64, 290
 link-editor support functions —
 ld_input_section, 290
 link-editor support functions —
 ld_section64, 290
 link-editor support functions — ld_section, 290
 link-editor support functions — ld_start64, 290
 link-editor support functions — ld_start, 290
 link-editor support functions —
 ld_support, 290
 link-editor support functions — ld_version, 290
 load exponent of a radix-independent floating-point number — scalb, 555
 load exponent of a radix-independent floating-point number — scalbn, 556
 log — natural logarithm function, 325
 log gamma function — gamma, 291
 log gamma function — gamma_r, 291
 log gamma function — lgamma, 291
 log gamma function — lgamma_r, 291
 log a message in system log — picld_log, 433
 log10 — base 10 logarithm function, 323
 log1p — compute natural logarithm, 324
 logb — radix-independent exponent, 326

M
 m_create_layout — initialize a layout object, 335
 m_destroy_layout — destroy a layout object, 339
 m_getvalues_layout — query layout values of a LayoutObject, 346
 m_setvalues_layout — set layout values of a LayoutObject, 351
 m_transform_layout — layout transformation, 352
 m_wtransform_layout — layout transformation for wide character strings, 357
 maillock — functions to manage lockfile(s) for user's mailbox, 327

- mailunlock — functions to manage lockfile(s) for user's mailbox, 327
- manage a name-value pair list — nvlist_alloc, 371
- manage a name-value pair list — nvlist_dup, 371
- manage a name-value pair list — nvlist_free, 371
- manage a name-value pair list — nvlist_pack, 371
- manage a name-value pair list — nvlist_size, 371
- manage a name-value pair list — nvlist_unpack, 371
- Management Interface database administration functions
 - DmiAddComponent, 118
 - DmiAddGroup, 118
 - DmiAddLanguage, 118
 - DmiDeleteComponent, 118
 - DmiDeleteGroup, 118
 - DmiDeleteLanguage, 118
- Management Interface initialization functions
 - DmiGetConfig, 128
 - DmiGetVersion, 128
 - DmiRegister, 128
 - DmiSetConfig, 128
 - DmiUnregister, 128
- Management Interface listing functions
 - DmiListAttributes, 131
 - DmiListClassNames, 131
 - DmiListComponents, 131
 - DmiListComponentsByClass, 131
 - DmiListGroups, 131
 - DmiListLanguages, 131
- Management Interface operation functions
 - DmiAddRow, 122
 - DmiDeleteRow, 122
 - DmiGetAttribute, 122
 - DmiGetMultiple, 122
 - DmiSetAttribute, 122
 - DmiSetMultiple, 122
- manipulate resource pool configurations — pool_conf_alloc, 458
- manipulate resource pool configurations — pool_conf_close, 458
- manipulate resource pool configurations — pool_conf_commit, 458
- manipulate resource pool configurations — pool_conf_export, 458
- manipulate resource pool configurations — pool_conf_free, 458
- manipulate resource pool configurations — pool_conf_info, 458
- manipulate resource pool configurations — pool_conf_location, 458
- manipulate resource pool configurations — pool_conf_open, 458
- manipulate resource pool configurations — pool_conf_remove, 458
- manipulate resource pool configurations — pool_conf_rollback, 458
- manipulate resource pool configurations — pool_conf_status, 458
- manipulate resource pool configurations — pool_conf_validate, 458
- map or unmap imported segment — rsm_memseg_import_map, 544
- map or unmap imported segment — rsm_memseg_import_unmap, 544
- map a tnftcl error code to a string — tnftcl_strerror, 648
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_boolean, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_byte, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_byte_array, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_int16, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_int16_array, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_int32, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_int32_array, 376
- match name and type indicated by the interface name and retrieve data value — nvlist_lookup_int64, 376

match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_int64_array, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_string, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_string_array, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint16, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint16_array, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint32, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint32_array, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint64, 376
 match name and type indicated by the interface
 name and retrieve data value —
 nvlist_lookup_uint64_array, 376
 match_execattr — get execution profile
 entry, 229
 math library exception-handling —
 matherr, 329
 matherr — math library exception-
 handling, 329
 md5 — MD5 digest functions, 337
 MD5 digest functions — md5, 337
 MD5 digest functions — MD5Final, 337
 MD5 digest functions — MD5Init, 337
 MD5 digest functions — MD5Update, 337
 MD5 digest functions — md5_calc, 337
 md5_calc — MD5 digest functions, 337
 MD5Final — MD5 digest functions, 337
 MD5Init — MD5 digest functions, 337
 MD5Update — MD5 digest functions, 337
 media_findname — convert a supplied name
 into an absolute pathname that can be used
 to access removable media, 340
 media_getattr — get and set media
 attributes, 343
 media_setattr — get and set media
 attributes, 343
 memory management, copy a file into memory
 — copylist, 57
 mkdirp — create directories in a path, 347
 modify/delete user credentials for an
 authentication service — pam_setcred, 401
 mp — multiple precision integer
 arithmetic, 349
 mp_gcd — multiple precision integer
 arithmetic, 349
 mp_itom — multiple precision integer
 arithmetic, 349
 mp_madd — multiple precision integer
 arithmetic, 349
 mp_mcmp — multiple precision integer
 arithmetic, 349
 mp_mdiv — multiple precision integer
 arithmetic, 349
 mp_mfree — multiple precision integer
 arithmetic, 349
 mp_min — multiple precision integer
 arithmetic, 349
 mp_mout — multiple precision integer
 arithmetic, 349
 mp_msub — multiple precision integer
 arithmetic, 349
 mp_mtox — multiple precision integer
 arithmetic, 349
 mp_mult — multiple precision integer
 arithmetic, 349
 mp_pow — multiple precision integer
 arithmetic, 349
 mp_rpow — multiple precision integer
 arithmetic, 349
 mp_xtom — multiple precision integer
 arithmetic, 349
 multiple precision integer arithmetic
 — mp, 349
 — mp_gcd, 349
 — mp_itom, 349
 — mp_madd, 349
 — mp_mcmp, 349
 — mp_mdiv, 349
 — mp_mfree, 349
 — mp_min, 349
 — mp_mout, 349
 — mp_msub, 349

multiple precision integer arithmetic
(Continued)

- mp_mtox, 349
- mp_mult, 349
- mp_pow, 349
- mp_rpow, 349
- mp_xtom, 349

N

natural logarithm function — log, 325
newDmiOctetString — create DmiOctetString in dynamic memory, 363
newDmiString — create DmiString in dynamic memory, 364
next representable double-precision floating-point number — nextafter, 365
nextafter — next representable double-precision floating-point number, 365
NOTE — annotate source code with info for tools, 367
_NOTE — annotate source code with info for tools, 367
NOTE — annotate source code with info for tools
NOTE vs _NOTE, 368
NoteInfo Argument, 368
nvlist_alloc — manage a name-value pair list, 371
nvlist_dup — manage a name-value pair list, 371
nvlist_free — manage a name-value pair list, 371
nvlist_lookup_boolean — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_byte — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_byte_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_int16 — match name and type indicated by the interface name and retrieve data value, 376

nvlist_lookup_int16_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_int32 — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_int32_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_int64 — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_int64_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_string — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_string_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint16 — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint16_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint32 — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint32_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint64 — match name and type indicated by the interface name and retrieve data value, 376
nvlist_lookup_uint64_array — match name and type indicated by the interface name and retrieve data value, 376
nvlist_next_nvpair — return data regarding name-value pairs, 378
nvlist_pack — manage a name-value pair list, 371
nvlist_size — manage a name-value pair list, 371
nvlist_unpack — manage a name-value pair list, 371

nvpair_name — return data regarding name-value pairs, 378
 nvpair_type — return data regarding name-value pairs, 378
 nvpair_value_byte — retrieve value from a name-value pair, 381
 nvpair_value_byte_array — retrieve value from a name-value pair, 381
 nvpair_value_int16 — retrieve value from a name-value pair, 381
 nvpair_value_int16_array — retrieve value from a name-value pair, 381
 nvpair_value_int32 — retrieve value from a name-value pair, 381
 nvpair_value_int32_array — retrieve value from a name-value pair, 381
 nvpair_value_int64 — retrieve value from a name-value pair, 381
 nvpair_value_int64_array — retrieve value from a name-value pair, 381
 nvpair_value_string — retrieve value from a name-value pair, 381
 nvpair_value_string_array — retrieve value from a name-value pair, 381
 nvpair_value_uint16 — retrieve value from a name-value pair, 381
 nvpair_value_uint16_array — retrieve value from a name-value pair, 381
 nvpair_value_uint32 — retrieve value from a name-value pair, 381
 nvpair_value_uint32_array — retrieve value from a name-value pair, 381
 nvpair_value_uint64 — retrieve value from a name-value pair, 381
 nvpair_value_uint64_array — retrieve value from a name-value pair, 381

O

open or close exact files — ea_close, 139
 open or close exact files — ea_open, 139

P

p2close — close pipes to and from a command, 383

p2open — open pipes to and from a command, 383
 PAM — PAM Service Module APIs, 407
 PAM — Pluggable Authentication Module, 385
 pam — Pluggable Authentication Module
 Administrative Interface, 387
 Interface Overview, 385
 Stacking Multiple Schemes, 386
 Stateful Interface, 386
 pam_acct_mgmt — perform PAM account validation procedures, 388
 pam_authenticate — perform authentication within the PAM framework, 389
 pam_chauthtok — perform password related functions within the PAM framework, 391
 pam_close_session — perform PAM session creation and termination operations, 397
 pam_end — authentication transaction routines for PAM, 422
 PAM error messages, get string —
 pam_strerror, 425
 pam_get_data — PAM routines to maintain module specific state, 403
 pam_get_item — authentication information routines for PAM, 405
 pam_getenv — returns the value for a PAM environment name, 393
 pam_getenvlist — returns a list of all the PAM environment variables, 394
 pam_open_session — perform PAM session creation and termination operations, 397
 pam_putenv — change or add a value to the PAM environment, 399
 PAM routines to maintain module specific state
 — pam_get_data, 403
 — pam_set_data, 403
 PAM Service Module APIs, — PAM, 407
 pam_set_data — PAM routines to maintain module specific state, 403
 pam_set_item — authentication information routines for PAM, 405
 pam_setcred — modify/delete user credentials for an authentication service, 401
 pam_sm — PAM Service Module APIs
 Interaction with the User, 408
 Interface Overview, 407
 Stateful Interface, 407

pam_sm_acct_mgmt — service provider implementation for pam_acct_mgmt, 411
 pam_sm_authenticate — service provider implementation for pam_authenticate, 413
 pam_sm_chauthtok — service provider implementation for pam_chauthtok, 415
 pam_sm_close_session — Service provider implementation for pam_open_session and pam_close_session, 418
 pam_sm_open_session — Service provider implementation for pam_open_session and pam_close_session, 418
 pam_sm_setcred — service provider implementation for pam_setcred, 420
 pam_start — authentication transaction routines for PAM, 422
 pathfind — search for named file in named directories, 426
 pctx_capture — process context library, 428
 pctx_create — process context library, 428
 pctx_release — process context library, 428
 pctx_run — process context library, 428
 pctx_set_events — associate callbacks with process events, 430
 perform mutex locking on a card — SCF_Card_lock, 559
 perform mutex locking on a card — SCF_Card_unlock, 559
 perform a reset of a smartcard — SCF_Card_reset, 562
 perform authentication within the PAM framework — pam_authenticate, 389
 perform PAM account validation procedures — pam_acct_mgmt, 388
 perform PAM session creation and termination operations
 — pam_close_session, 397
 — pam_open_session, 397
 perform password related functions within the PAM framework — pam_chauthtok, 391
 picl_get_first_prop — get a property handle of a node, 436
 picl_get_next_by_col — access a table property, 438
 picl_get_next_by_row — access a table property, 438
 picl_get_next_prop — get a property handle of a node, 436
 picl_get_prop_by_name — get the handle of the property by name, 439
 picl_get_propinfo — get the information about a property, 440
 picl_get_propinfo_by_name — get property information and handle of named property, 441
 picl_get_propval — get the value of a property, 442
 picl_get_propval_by_name — get the value of a property, 442
 picl_get_root — get the root handle of the PICL tree, 444
 picl_initialize — initiate a session with the PICL daemon, 445
 PICL interface library — libpicl, 312
 picl_set_propval — set the value of a property to the specified value, 446
 picl_set_propval_by_name — set the value of a property to the specified value, 446
 picl_shutdown — shutdown the session with the PICL daemon, 448
 picl_strerror — get error message string, 449
 picl_wait — wait for PICL tree to refresh, 450
 picl_walk_tree_by_class — walk subtree by class, 451
 picld_log — log a message in system log, 433
 picld_plugin_register — register plug-in with the daemon, 434
 pipes, open, close to and from a command — p2open, p2close, 383
 place process in new project with attendant resource controls, resource pools, and attributes — setproject, 595
 Pluggable Authentication Module, — PAM, 385
 pool_associate — resource pool manipulation functions, 452
 pool_component_info — resource pool component functions, 455
 pool_component_to_elem — resource pool element-related functions, 457
 pool_conf_alloc — manipulate resource pool configurations, 458
 pool_conf_close — manipulate resource pool configurations, 458
 pool_conf_commit — manipulate resource pool configurations, 458

pool_conf_export — manipulate resource pool configurations, 458
 pool_conf_free — manipulate resource pool configurations, 458
 pool_conf_info — manipulate resource pool configurations, 458
 pool_conf_location — manipulate resource pool configurations, 458
 pool_conf_open — manipulate resource pool configurations, 458
 pool_conf_remove — manipulate resource pool configurations, 458
 pool_conf_rollback — manipulate resource pool configurations, 458
 pool_conf_status — manipulate resource pool configurations, 458
 pool_conf_to_elem — resource pool element-related functions, 457
 pool_conf_validate — manipulate resource pool configurations, 458
 pool_create — resource pool manipulation functions, 452
 pool_destroy — resource pool manipulation functions, 452
 pool_dissociate — resource pool manipulation functions, 452
 pool_dynamic_location — resource pool framework functions, 463
 pool_error — error interface to resource pools library, 465
 pool_get_binding — set and query process to resource pool bindings, 467
 pool_get_owning_resource — resource pool component functions, 455
 pool_get_pool — retrieve resource pool configuration elements, 470
 pool_get_property — resource pool element property manipulation, 472
 pool_get_resource — retrieve resource pool configuration elements, 470
 pool_get_resource_binding — set and query process to resource pool bindings, 467
 pool_info — resource pool manipulation functions, 452
 pool_put_property — resource pool element property manipulation, 472
 pool_query_components — retrieve resource pool configuration elements, 470
 pool_query_pool_resources — resource pool manipulation functions, 452
 pool_query_pools — retrieve resource pool configuration elements, 470
 pool_query_resource_components — resource pool resource manipulation functions, 475
 pool_query_resources — retrieve resource pool configuration elements, 470
 pool_resource_create — resource pool resource manipulation functions, 475
 pool_resource_destroy — resource pool resource manipulation functions, 475
 pool_resource_info — resource pool resource manipulation functions, 475
 pool_resource_to_elem — resource pool element-related functions, 457
 pool_resource_transfer — resource pool resource manipulation functions, 475
 pool_resource_xtransfer — resource pool resource manipulation functions, 475
 pool_rm_property — resource pool element property manipulation, 472
 pool_set_binding — set and query process to resource pool bindings, 467
 pool_static_location — resource pool framework functions, 463
 pool_strerror — error interface to resource pools library, 465
 pool_to_elem — resource pool element-related functions, 457
 pool_value_alloc — resource pool property value manipulation functions, 479
 pool_value_free — resource pool property value manipulation functions, 479
 pool_value_get_bool — resource pool property value manipulation functions, 479
 pool_value_get_double — resource pool property value manipulation functions, 479
 pool_value_get_int64 — resource pool property value manipulation functions, 479
 pool_value_get_name — resource pool property value manipulation functions, 479
 pool_value_get_string — resource pool property value manipulation functions, 479
 pool_value_get_type — resource pool property value manipulation functions, 479
 pool_value_get_uint64 — resource pool property value manipulation functions, 479

pool_value_set_bool — resource pool property value manipulation functions, 479
 pool_value_set_double — resource pool property value manipulation functions, 479
 pool_value_set_int64 — resource pool property value manipulation functions, 479
 pool_value_set_name — resource pool property value manipulation functions, 479
 pool_value_set_string — resource pool property value manipulation functions, 479
 pool_value_set_uint64 — resource pool property value manipulation functions, 479
 pool_version — resource pool framework functions, 463
 pool_walk_components — walk objects within resource pool configurations, 482
 pool_walk_pools — walk objects within resource pool configurations, 482
 pool_walk_properties — resource pool element property manipulation, 472
 pool_walk_resources — walk objects within resource pool configurations, 482
 post a PICL event — ptree_post_event, 510
 pow — power function, 484
 power function — pow, 484
 power up or power down the smart card — IFDHPowerICC, 259
 print a DmiString, — printDmiString, 487
 print data in DmiAttributeValues list, — printDmiAttributeValues, 485
 print data in input data union, — printDmiDataUnion, 486
 print error in string form, — dmi_error, 127
 printDmiAttributeValues— print data in DmiAttributeValues list, 485
 printDmiDataUnion— print data in input data union, 486
 printDmiString— print a DmiString, 487
 probe insertion interface
 — TNF_DEBUG, 656
 — TNF_PROBE_0, 656
 — TNF_PROBE_0_DEBUG, 656
 — TNF_PROBE_1, 656
 — TNF_PROBE_1_DEBUG, 656
 — TNF_PROBE_2, 656
 — TNF_PROBE_2_DEBUG, 656
 — TNF_PROBE_3, 656
 — TNF_PROBE_3_DEBUG, 656
 probe insertion interface (Continued)
 — TNF_PROBE_4, 656
 — TNF_PROBE_4_DEBUG, 656
 — TNF_PROBE_5, 656
 — TNF_PROBE_5_DEBUG, 656
 process context library — pctx_capture, 428
 process context library — pctx_create, 428
 process context library — pctx_release, 428
 process context library — pctx_run, 428
 project database entry functions — endproject, 235
 project database entry functions — fgetproject, 235
 project database entry functions — getdefaultproj, 235
 project database entry functions — getprojbyid, 235
 project database entry functions — getprojbyname, 235
 project database entry functions — getproject, 235
 project database entry functions — inproj, 235
 project database entry functions — setproject, 235
 project_walk — visit active project IDs on current system, 488
 provide a transient program number, — reg_ci_callback, 516
 ptree_add_node — add or delete node to or from tree, 490
 PTree and Plug-in Registration interface library — libpicltree, 315
 ptree_create_and_add_node — create and add node to tree and return node handle, 492
 ptree_create_and_add_prop — create and add property to node and return property handle, 493
 ptree_create_prop — create or destroy a property, 496
 ptree_delete_node — add or delete node to or from tree, 490
 ptree_destroy_prop — create or destroy a property, 496
 ptree_find_node — find node with given property and value, 499
 ptree_get_node_by_path — get handle of node specified by PICL tree path, 502

ptree_get_propinfo — get property information, 505
 ptree_get_propinfo_by_name — get property information and handle of named property, 506
 ptree_get_propval — get the value of a property, 507
 ptree_get_propval_by_name — get the value of a property, 507
 ptree_get_root — get the root node handle, 508
 ptree_init_propinfo — initialize ptree_propinfo_t structure, 509
 ptree_post_event — post a PICL event, 510
 ptree_register_handler — register a handler for the event, 511
 ptree_unregister_handler — unregister the event handler for the event, 512
 ptree_update_propval — update a property value, 513
 ptree_update_propval_by_name — update a property value, 513
 ptree_walk_tree_by_class — walk subtree by class, 514

Q

query layout values of a LayoutObject — m_getvalues_layout, 346

R

radix-independent exponent — logb, 326
 rd_delete — runtime linker debugging functions, 554
 rd_errstr — runtime linker debugging functions, 554
 rd_event_addr — runtime linker debugging functions, 554
 rd_event_enable — runtime linker debugging functions, 554
 rd_event_getmsg — runtime linker debugging functions, 554
 rd_init — runtime linker debugging functions, 554
 rd_loadobj_iter — runtime linker debugging functions, 554

rd_log — runtime linker debugging functions, 554
 rd_new — runtime linker debugging functions, 554
 rd_objpad_enable — runtime linker debugging functions, 554
 rd_plt_resolution — runtime linker debugging functions, 554
 rd_reset — runtime linker debugging functions, 554
 read and write a disk's VTOC — read_vtoc, 515
 read from a segment —
 rsm_memseg_import_get16, 541
 read from a segment —
 rsm_memseg_import_get32, 541
 read from a segment —
 rsm_memseg_import_get64, 541
 read from a segment —
 rsm_memseg_import_get8, 541
 read from a segment —
 rsm_memseg_import_get, 541
 read system process structures
 — kvm_getproc, 283
 — kvm_nextproc, 283
 — kvm_setproc, 283
 read and write a disk's VTOC — read_vtoc, write_vtoc, 515
 read or write kstat data
 — kstat_read, 279
 — kstat_write, 279
 read_vtoc — read and write a disk's VTOC, 515
 receive asynchronous event notification — SCF_Terminal_addEventListener, 578
 receive asynchronous event notification — SCF_Terminal_removeEventListener, 578
 receive asynchronous event notification — SCF_Terminal_updateEventListener, 578
 regexpr — regular expression compile and match routines, 517
 register a component in the product install registry — wsreg_register, 703
 register a handler for the event — ptree_register_handler, 511
 register callbacks for probe creation and destruction — tnftcl_register_funcs, 647
 register plug-in with the daemon — picld_plugin_register, 434

- regular expression compile and match routines
 - advance, 517
 - compile, 517
 - regexpr, 517
 - step, 517
- release removable media device reservation —
 - volmgt_release, 675
- remainder — remainder function, 520
- remainder function — remainder, 520
- remote memory access error detection functions
 - rsm_memseg_import_close_barrier, 546
- remote memory access error detection functions
 - rsm_memseg_import_open_barrier, 546
- remote memory access error detection functions
 - rsm_memseg_import_order_barrier, 546
- remove a component from the product install
 - registry — wsreg_unregister, 717
- reserve removable media device —
 - volmgt_acquire, 667
- resource allocation and management functions
 - for export memory segments —
 - rsm_memseg_export_create, 532
- resource allocation and management functions
 - for export memory segments —
 - rsm_memseg_export_destroy, 532
- resource allocation and management functions
 - for export memory segments —
 - rsm_memseg_export_rebind, 532
- resource pool component functions —
 - pool_component_info, 455
- resource pool component functions —
 - pool_get_owning_resource, 455
- resource pool element property manipulation —
 - pool_get_property, 472
- resource pool element property manipulation —
 - pool_put_property, 472
- resource pool element property manipulation —
 - pool_rm_property, 472
- resource pool element property manipulation —
 - pool_walk_properties, 472
- resource pool element-related functions —
 - pool_component_to_elem, 457
- resource pool element-related functions —
 - pool_conf_to_elem, 457
- resource pool element-related functions —
 - pool_resource_to_elem, 457
- resource pool element-related functions —
 - pool_to_elem, 457
- resource pool framework functions —
 - pool_dynamic_location, 463
- resource pool framework functions —
 - pool_static_location, 463
- resource pool framework functions —
 - pool_version, 463
- resource pool manipulation functions —
 - pool_associate, 452
- resource pool manipulation functions —
 - pool_create, 452
- resource pool manipulation functions —
 - pool_destroy, 452
- resource pool manipulation functions —
 - pool_dissociate, 452
- resource pool manipulation functions —
 - pool_info, 452
- resource pool manipulation functions —
 - pool_query_pool_resources, 452
- resource pool property value manipulation
 - functions — pool_value_alloc, 479
- resource pool property value manipulation
 - functions — pool_value_free, 479
- resource pool property value manipulation
 - functions — pool_value_get_bool, 479
- resource pool property value manipulation
 - functions — pool_value_get_double, 479
- resource pool property value manipulation
 - functions — pool_value_get_int64, 479
- resource pool property value manipulation
 - functions — pool_value_get_name, 479
- resource pool property value manipulation
 - functions — pool_value_get_string, 479
- resource pool property value manipulation
 - functions — pool_value_get_type, 479
- resource pool property value manipulation
 - functions — pool_value_get_uint64, 479
- resource pool property value manipulation
 - functions — pool_value_set_bool, 479
- resource pool property value manipulation
 - functions — pool_value_set_double, 479
- resource pool property value manipulation
 - functions — pool_value_set_int64, 479
- resource pool property value manipulation
 - functions — pool_value_set_name, 479
- resource pool property value manipulation
 - functions — pool_value_set_string, 479
- resource pool property value manipulation
 - functions — pool_value_set_uint64, 479

- resource pool resource manipulation functions
 - pool_query_resource_components, 475
- resource pool resource manipulation functions
 - pool_resource_create, 475
- resource pool resource manipulation functions
 - pool_resource_destroy, 475
- resource pool resource manipulation functions
 - pool_resource_info, 475
- resource pool resource manipulation functions
 - pool_resource_transfer, 475
- resource pool resource manipulation functions
 - pool_resource_xtransfer, 475
- retrieve information about a session, terminal, or card — SCF_Card_getInfo, 568
- retrieve information about a session, terminal, or card — SCF_Session_getInfo, 568
- retrieve information about a session, terminal, or card — SCF_Terminal_getInfo, 568
- retrieve resource pool configuration elements — pool_get_pool, 470
- retrieve resource pool configuration elements — pool_get_resource, 470
- retrieve resource pool configuration elements — pool_query_components, 470
- retrieve resource pool configuration elements — pool_query_pools, 470
- retrieve resource pool configuration elements — pool_query_resources, 470
- retrieve value from a name-value pair — nvpair_value_byte, 381
- retrieve value from a name-value pair — nvpair_value_byte_array, 381
- retrieve value from a name-value pair — nvpair_value_int16, 381
- retrieve value from a name-value pair — nvpair_value_int16_array, 381
- retrieve value from a name-value pair — nvpair_value_int32, 381
- retrieve value from a name-value pair — nvpair_value_int32_array, 381
- retrieve value from a name-value pair — nvpair_value_int64, 381
- retrieve value from a name-value pair — nvpair_value_int64_array, 381
- retrieve value from a name-value pair — nvpair_value_string, 381
- retrieve value from a name-value pair — nvpair_value_string_array, 381
- retrieve value from a name-value pair — nvpair_value_uint16, 381
- retrieve value from a name-value pair — nvpair_value_uint16_array, 381
- retrieve value from a name-value pair — nvpair_value_uint32, 381
- retrieve value from a name-value pair — nvpair_value_uint32_array, 381
- retrieve value from a name-value pair — nvpair_value_uint64, 381
- retrieve value from a name-value pair — nvpair_value_uint64_array, 381
- retrieve archive symbol table — elf_getarsym, 180
- retrieve class-dependent object file header
 - elf32_getehdr, 153
 - elf32_newehdr, 153
 - elf64_getehdr, 153
 - elf64_newehdr, 153
- retrieve class-dependent program header table
 - elf32_getphdr, 155
 - elf32_newphdr, 155
 - elf64_getphdr, 155
 - elf64_newphdr, 155
- retrieve class-dependent section header
 - elf32_getshdr, 157
 - elf64_getshdr, 157
- return data regarding name-value pairs — nvlist_next_nvpair, 378
- return data regarding name-value pairs — nvpair_name, 378
- return data regarding name-value pairs — nvpair_type, 378
- return libdevinfo minor node information — di_minor_devt, 101
- return libdevinfo minor node information — di_minor_name, 101
- return libdevinfo minor node information — di_minor_nodetype, 101
- return libdevinfo minor node information — di_minor_spectype, 101
- returns a list of all the PAM environment variables — pam_getenvlist, 394
- return magnitude of first argument and sign of second argument — copysign, 58
- return root lgroup ID — lgrp_root, 305
- return the size of an object file type
 - elf32_fsize, 152

return the size of an object file type (Continued)
 — `elf64_fsize`, 152
 returns the value for a PAM environment name
 — `pam_getenv`, 393
 return the Volume Management root directory
 — `volmgt_root`, 676
 return whether or not Volume Management is
 running — `volmgt_running`, 677
 returns an unbiased exponent — `ilogb`, 266
 rint — round-to-nearest integral value, 521
 rmdirp — remove directories in a path, 347
 round-to-nearest integral value — `rint`, 521
`rsm_create_localmemory_handle` — create or
 free local memory handle, 522
`rsm_free_interconnect_topology` — get or free
 interconnect topology, 526
`rsm_free_localmemory_handle` — create or free
 local memory handle, 522
`rsm_get_controller` — get or release a controller
 handle, 524
`rsm_get_controller_attr` — get or release a
 controller handle, 524
`rsm_get_interconnect_topology` — get or free
 interconnect topology, 526
`rsm_get_segmentid_range` — get segment ID
 range, 528
`rsm_intr_signal_post` — signal or wait for an
 event, 530
`rsm_intr_signal_wait` — signal or wait for an
 event, 530
`rsm_memseg_export_create` — resource
 allocation and management functions for
 export memory segments, 532
`rsm_memseg_export_destroy` — resource
 allocation and management functions for
 export memory segments, 532
`rsm_memseg_export_publish` — allow or
 disallow a memory segment to be imported
 by other nodes, 535
`rsm_memseg_export_rebind` — resource
 allocation and management functions for
 export memory segments, 532
`rsm_memseg_export_republish` — allow or
 disallow a memory segment to be imported
 by other nodes, 535
`rsm_memseg_export_unpublish` — allow or
 disallow a memory segment to be imported
 by other nodes, 535
`rsm_memseg_get_pollfd` — get or release a poll
 descriptor, 538
`rsm_memseg_import_close_barrier` — remote
 memory access error detection
 functions, 546
`rsm_memseg_import_connect` — create or break
 logical connection between import and
 export segments, 539
`rsm_memseg_import_destroy_barrier` — create
 or destroy barrier for imported segment, 543
`rsm_memseg_import_disconnect` — create or
 break logical connection between import
 and export segments, 539
`rsm_memseg_import_get` — read from a
 segment, 541
`rsm_memseg_import_get_mode` — set or get
 mode for barrier scoping, 552
`rsm_memseg_import_get16` — read from a
 segment, 541
`rsm_memseg_import_get32` — read from a
 segment, 541
`rsm_memseg_import_get64` — read from a
 segment, 541
`rsm_memseg_import_get8` — read from a
 segment, 541
`rsm_memseg_import_getv` — write to a
 segment using a list of I/O requests, 550
`rsm_memseg_import_init_barrier` — create or
 destroy barrier for imported segment, 543
`rsm_memseg_import_map` — map or unmap
 imported segment, 544
`rsm_memseg_import_open_barrier` — remote
 memory access error detection
 functions, 546
`rsm_memseg_import_order_barrier` — remote
 memory access error detection
 functions, 546
`rsm_memseg_import_put` — write to a
 segment, 548
`rsm_memseg_import_put16` — write to a
 segment, 548
`rsm_memseg_import_put32` — write to a
 segment, 548
`rsm_memseg_import_put64` — write to a
 segment, 548
`rsm_memseg_import_put8` — write to a
 segment, 548

- rsm_memseg_import_putv — write to a segment using a list of I/O requests, 550
- rsm_memseg_import_set_mode — set or get mode for barrier scoping, 552
- rsm_memseg_import_unmap — map or unmap imported segment, 544
- rsm_memseg_release_pollfd — get or release a poll descriptor, 538
- rsm_release_controller — get or release a controller handle, 524
- rtld_audit — runtime linker auditing functions, 553
- rtld_db — runtime linker debugging functions, 554
- runtime linker auditing functions — la_activity, 553
- runtime linker auditing functions — la_i86_pltenter, 553
- runtime linker auditing functions — la_objopen, 553
- runtime linker auditing functions — la_objsearch, 553
- runtime linker auditing functions — la_pltexit64, 553
- runtime linker auditing functions — la_pltexit, 553
- runtime linker auditing functions — la_preinit, 553
- runtime linker auditing functions — la_sparcv8_pltenter, 553
- runtime linker auditing functions — la_sparcv9_pltenter, 553
- runtime linker auditing functions — la_symbind32, 553
- runtime linker auditing functions — la_symbind64, 553
- runtime linker auditing functions — la_version, 553
- runtime linker auditing functions — rtld_audit, 553
- runtime linker debugging functions — rd_delete, 554
- runtime linker debugging functions — rd_errstr, 554
- runtime linker debugging functions — rd_event_addr, 554
- runtime linker debugging functions — rd_event_enable, 554

- runtime linker debugging functions — rd_event_getmsg, 554
- runtime linker debugging functions — rd_init, 554
- runtime linker debugging functions — rd_loadobj_iter, 554
- runtime linker debugging functions — rd_log, 554
- runtime linker debugging functions — rd_new, 554
- runtime linker debugging functions — rd_objpad_enable, 554
- runtime linker debugging functions — rd_plt_resolution, 554
- runtime linker debugging functions — rd_reset, 554
- runtime linker debugging functions — rtld_db, 554

S

- scalb — load exponent of a radix-independent floating-point number, 555
- scalbn — load exponent of a radix-independent floating-point number, 556
- SCF_Card_exchangeAPDU — send a command APDU to a card and read the card's response, 557
- SCF_Card_getInfo — retrieve information about a session, terminal, or card, 568
- SCF_Card_lock — perform mutex locking on a card, 559
- SCF_Card_reset — perform a reset of a smartcard, 562
- SCF_Card_unlock — perform mutex locking on a card, 559
- SCF_Card_waitForCardRemoved — wait for a card to be inserted or removed, 586
- SCF_Session_close — close a smartcard session, terminal, or card, 564
- SCF_Session_getInfo — retrieve information about a session, terminal, or card, 568
- SCF_Session_getSession — establish a context with a system's smartcard framework, 572
- SCF_Session_getTerminal — establish a context with a smartcard terminal (reader), 574

SCF_strerror — get a string describing a status code, 577

SCF_Terminal_addEventListener — receive asynchronous event notification, 578

SCF_Terminal_close — close a smartcard session, terminal, or card, 564

SCF_Terminal_getCard — establish a context with a smartcard, 584

SCF_Terminal_getInfo — retrieve information about a session, terminal, or card, 568

SCF_Terminal_removeEventListener — receive asynchronous event notification, 578

SCF_Terminal_updateEventListener — receive asynchronous event notification, 578

SCF_Terminal_waitForCardAbsent — wait for a card to be inserted or removed, 586

SCF_Terminal_waitForCardPresent — wait for a card to be inserted or removed, 586

search for a property —

- di_prop_lookup_bytes, 111

search for a property —

- di_prop_lookup_int64, 111

search for a property —

- di_prop_lookup_ints, 111

search for a property —

- di_prop_lookup_strings, 111

send a command APDU to a card and read the card's response —

- SCF_Card_exchangeAPDU, 557

send a file — sendfilev, 592

send control information to an IFD —

- IFDHControl, 252

send files over sockets or copy files to files —

- sendfile, 589

sendfile — send files over sockets or copy files to files, 589

sendfilev — send a file, 592

Service Provider functions for components

- DmiOriginateEvent, 136
- DmiRegisterCi, 136
- DmiUnRegisterCi, 136

service provider implementation for

- pam_acct_mgmt —
- pam_sm_acct_mgmt, 411

service provider implementation for

- pam_authenticate —
- pam_sm_authenticate, 413

service provider implementation for

- pam_chauthtok — pam_sm_chauthtok, 415

Service provider implementation for

- pam_open_session and pam_close_session —
- pam_sm_close_session, 418
- pam_sm_open_session, 418

service provider implementation for

- pam_setcred — pam_sm_setcred, 420

set and query process to resource pool bindings —

- pool_get_binding, 467

set and query process to resource pool bindings —

- pool_get_resource_binding, 467

set and query process to resource pool bindings —

- pool_set_binding, 467

set or get mode for barrier scoping —

- rsm_memseg_import_get_mode, 552

set or get mode for barrier scoping —

- rsm_memseg_import_set_mode, 552

set or get the instance of a component —

- wsreg_get_instance, 708

set or get the instance of a component —

- wsreg_set_instance, 708

set or get the instance of a query —

- wsreg_query_get_instance, 699

set or get the instance of a query —

- wsreg_query_set_instance, 699

set or get the location of a component —

- wsreg_get_location, 710

set or get the location of a component —

- wsreg_set_location, 710

set or get the location of a query —

- wsreg_query_get_location, 700

set or get the location of a query —

- wsreg_query_set_location, 700

set or get the parent of a component —

- wsreg_get_parent, 711

set or get the parent of a component —

- wsreg_set_parent, 711

set or get the type of a component —

- wsreg_get_type, 712

set or get the type of a component —

- wsreg_set_type, 712

set or get the uninstaller of a component —

- wsreg_get_uninstaller, 713

set or get the uninstaller of a component —

- wsreg_set_uninstaller, 713

set or get the unique name of a component —

- wsreg_get_unique_name, 714

set or get the unique name of a component —
 wsreg_set_unique_name, 714
 set or get the unique name of a query —
 wsreg_query_get_unique_name, 701
 set or get the unique name of a query —
 wsreg_query_set_unique_name, 701
 set or get the uuid of a component —
 wsreg_get_id, 707
 set or get the uuid of a component —
 wsreg_set_id, 707
 set or get the uuid of a query —
 wsreg_query_get_id, 698
 set or get the uuid of a query —
 wsreg_query_set_id, 698
 set or get the vendor of a componentt —
 wsreg_get_vendor, 715
 set or get the vendor of a componentt —
 wsreg_set_vendor, 715
 set or get the version of a component —
 wsreg_get_version, 716
 set or get the version of a component —
 wsreg_set_version, 716
 set or get the version of a query —
 wsreg_query_get_version, 702
 set or get the version of a query —
 wsreg_query_set_version, 702
 set the value of a property to the specified value
 — picl_set_propval, 446
 set the value of a property to the specified value
 — picl_set_propval_by_name, 446
 set layout values of a LayoutObject —
 m_setvalues_layout, 351
 set protocol parameters —
 IFDHSetProtocolParameters, 262
 set slot or card capabilities —
 IFDHSetCapabilities, 261
 setac — get audit control file information, 213
 setaclass — rewind audit_class database
 file, 215
 setauuser — rewind audit_event database
 file, 218
 setauthattr — get authorization database
 entry, 220
 setauuser — get audit_user database entry, 223
 setddent — get device_deallocate entry, 225
 setddfile — get device_deallocate entry, 225
 setdmapent — get device_maps entry, 227
 setdmapfile — get device_maps entry, 227
 setexecattr — get execution profile entry, 229
 setprofattr — get profile description and
 attributes, 233
 setproject — place process in new project with
 attendant resource controls, resource pools,
 and attributes, 595
 setprojent — project database entry
 functions, 235
 setuserattr — get user_attr entry, 239
 shell global pattern matching — gmatch, 241
 shutdown the session with the PICL daemon —
 picl_shutdown, 448
 signal or wait for an event —
 rsm_intr_signal_post, 530
 signal or wait for an event —
 rsm_intr_signal_wait, 530
 significand — significand function, 597
 significand function — significand, 597
 simple difference and accumulate operations —
 cpc_event_accum, 75
 simple difference and accumulate operations —
 cpc_event_diff, 75
 sin — sine function, 598
 sine function — sin, 598
 sinh — hyperbolic sine function, 599
 sort an ACL — aclsort, 24
 sqrt — square root function, 600
 square root function — sqrt, 600
 SSAAgentIsAlive — Sun Solstice Enterprise
 Agent registration and communication
 helper functions, 601
 SSAGetTrapPort — Sun Solstice Enterprise
 Agent registration and communication
 helper functions, 601
 SSAOidCmp — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidCpy — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidDup — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidFree — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidInit — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidNew — Sun Solstice Enterprise Agent
 OID helper functions, 604
 SSAOidString — Sun Solstice Enterprise Agent
 OID helper functions, 604

SSAOidStrToOid — Sun Solstice Enterprise Agent OID helper functions, 604

SSAOidZero — Sun Solstice Enterprise Agent OID helper functions, 604

SSARegSubagent — Sun Solstice Enterprise Agent registration and communication helper functions, 601

SSARegSubtable — Sun Solstice Enterprise Agent registration and communication helper functions, 601

SSARegSubtree — Sun Solstice Enterprise Agent registration and communication helper functions, 601

SSASendTrap — Sun Solstice Enterprise Agent registration and communication helper functions, 601

SSAStringCpy — Sun Solstice Enterprise Agent string helper functions, 606

SSAStringInit — Sun Solstice Enterprise Agent string helper functions, 606

SSAStringToChar — Sun Solstice Enterprise Agent string helper functions, 606

SSAStringZero — Sun Solstice Enterprise Agent string helper functions, 606

SSASubagentOpen — Sun Solstice Enterprise Agent registration and communication helper functions, 601

step — regular expression compile and match routines, 517

strcadd — copy strings, compressing or expanding C language escape codes, 607

strccpy — copy strings, compressing or expanding C language escape codes, 607

streadd — copy strings, compressing or expanding C language escape codes, 607

STREAMS

- determine whether a buffer of characters is encrypted — isencrypt, 267
- read stream up to next delimiter — bgets, 43
- split buffer into fields — bufsplit, 45

strecpy — copy strings, compressing or expanding C language escape codes, 607

strfind — string manipulations, 609

string manipulations — strfind, 609

- strrspn, 609
- strtrns, 609

string manipulations

- strfind, 609

string manipulations (Continued)

- strrspn, 609
- strtrns, 609

string operation, get PAM error message string — pam_strerror, 425

strings, copy, compressing or expanding C language escape codes, 607

Sun Solstice Enterprise Agent OID helper functions

- SSAOidCmp, 604
- SSAOidCpy, 604
- SSAOidDup, 604
- SSAOidFree, 604
- SSAOidInit, 604
- SSAOidNew, 604
- SSAOidString, 604
- SSAOidStrToOid, 604
- SSAOidZero, 604

Sun Solstice Enterprise Agent registration and communication helper functions

- SSAAgentIsAlive, 601
- SSAGetTrapPort, 601
- SSARegSubagent, 601
- SSARegSubtable, 601
- SSARegSubtree, 601
- SSASendTrap, 601
- SSASubagentOpen, 601

Sun Solstice Enterprise Agent string helper functions

- SSAStringCpy, 606
- SSAStringInit, 606
- SSAStringToChar, 606
- SSAStringZero, 606

sysevent_bind_handle — bind or unbind subscriber handle, 610

sysevent_free — free memory for sysevent handle, 612

sysevent_get_attr_list — get attribute list pointer, 613

sysevent_get_class_name — get class name, subclass name, ID or buffer size of event, 614

sysevent_get_event_id — get class name, subclass name, ID or buffer size of event, 614

sysevent_get_pid — get vendor name, publisher name or processor ID of event, 616

sysevent_get_pub_name — get vendor name,
 publisher name or processor ID of
 event, 616
 sysevent_get_size — get class name, subclass
 name, ID or buffer size of event, 614
 sysevent_get_subclass_name — get class name,
 subclass name, ID or buffer size of
 event, 614
 sysevent_get_vendor_name — get vendor
 name, publisher name or processor ID of
 event, 616
 sysevent_unbind_handle — bind or unbind
 subscriber handle, 610

T

tan — tangent function, 623
 tangent function — tan, 623
 tanh — hyperbolic tangent function, 624
 test access CPU performance counters —
 cpc_access, 64
 test for NaN — isnan, 268
 TNF_DEBUG — probe insertion interface, 656
 TNF_PROBE — probe insertion interface
 arg_name_n, 659
 arg_type_n, 658
 arg_value_n, 659
 detail, 657
 keys, 657
 name, 657
 TNF_PROBE_0 — probe insertion
 interface, 656
 TNF_PROBE_0_DEBUG — probe insertion
 interface, 656
 TNF_PROBE_1 — probe insertion
 interface, 656
 TNF_PROBE_1_DEBUG — probe insertion
 interface, 656
 TNF_PROBE_2 — probe insertion
 interface, 656
 TNF_PROBE_2_DEBUG — probe insertion
 interface, 656
 TNF_PROBE_3 — probe insertion
 interface, 656
 TNF_PROBE_3_DEBUG — probe insertion
 interface, 656

TNF_PROBE_4 — probe insertion
 interface, 656
 TNF_PROBE_4_DEBUG — probe insertion
 interface, 656
 TNF_PROBE_5 — probe insertion
 interface, 656
 TNF_PROBE_5_DEBUG — probe insertion
 interface, 656
 tnfn_process_disable() — disables probing for the
 process, 661
 tnfn_process_enable() — enables probing for the
 process, 661
 tnfn_thread_disable() — disables probing for the
 calling thread, 661
 tnfn_thread_enable() — enables probing for the
 calling thread, 661
 tnfnctl_buffer_alloc — allocate or deallocate a
 buffer for trace data, 625
 tnfnctl_buffer_dealloc — allocate or deallocate a
 buffer for trace data, 625
 tnfnctl_check_libs — control probes of another
 process where caller provides /proc
 functionality, 629
 tnfnctl_close — close a tnfnctl handle, 627
 tnfnctl_continue — interfaces for direct probe
 and process control for another process, 635
 tnfnctl_exec_open — interfaces for direct probe
 and process control for another process, 635
 tnfnctl_filter_list_add — control kernel tracing
 and process filtering, 651
 tnfnctl_filter_list_delete — control kernel tracing
 and process filtering, 651
 tnfnctl_filter_list_get — control kernel tracing
 and process filtering, 651
 tnfnctl_filter_state_set — control kernel tracing
 and process filtering, 651
 tnfnctl_indirect_open — control probes of
 another process where caller provides /proc
 functionality, 629
 tnfnctl_internal_open — create handle for
 internal process probe control, 632
 tnfnctl_kernel_open — create handle for kernel
 probe control, 634
 tnfnctl_pid_open — interfaces for direct probe
 and process control for another process, 635
 tnfnctl_probe_apply — iterate over probes, 640
 tnfnctl_probe_apply_ids — iterate over
 probes, 640

tnftcl_probe_connect — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_disable — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_disconnect_all — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_enable — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_state_get — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_trace — interfaces to query and to change the state of a probe, 643
 tnftcl_probe_untrace — interfaces to query and to change the state of a probe, 643
 tnftcl_register_funcs — register callbacks for probe creation and destruction, 647
 tnftcl_strerror — map a tnftcl error code to a string, 648
 tnftcl_trace_attrs_get — get the trace attributes from a tnftcl handle, 649
 tnftcl_trace_state_set — control kernel tracing and process filtering, 651
 touchlock — functions to manage lockfile(s) for user's mailbox, 327
 translate strings to and from events —
 cpc_eventtostr, 84
 translate strings to and from events —
 cpc_strtoevent, 84
 transmit APDU to a smart card —
 IFDHTransmitToICC, 264
 traverse libdevinfo device nodes —
 di_walk_node, 117
 traverse libdevinfo minor nodes —
 di_walk_minor, 115

U

unregister the event handler for the event —
 ptree_unregister_handler, 512
 update a property value —
 ptree_update_propval, 513
 update a property value —
 ptree_update_propval_by_name, 513
 use CPU performance counters on lwps —
 cpc_bind_event, 65
 use CPU performance counters on lwps —
 cpc_rele, 65

use CPU performance counters on lwps —
 cpc_take_sample, 65

V

visit active project IDs on current system —
 project_walk, 488
 volmgt_acquire — reserve removable media device, 667
 volmgt_check — have Volume Management check for media, 670
 volmgt_feature_enabled — check whether specific Volume Management features are enabled, 672
 volmgt_inuse — check whether or not Volume Management is managing a pathname, 673
 volmgt_release — release removable media device reservation, 675
 volmgt_root — return the Volume Management root directory, 676
 volmgt_running — return whether or not Volume Management is running, 677
 volmgt_symdev — convert between Volume Management symbolic names, and the devices that correspond to them, 678
 volmgt_symname — convert between Volume Management symbolic names, and the devices that correspond to them, 678
 VTOC, disk's
 read a disk's VTOC — read_vtoc, 515
 write a disk's VTOC — write_vtoc, 515

W

wait for a card to be inserted or removed —
 SCF_Card_waitForCardRemoved, 586
 wait for a card to be inserted or removed —
 SCF_Terminal_waitForCardAbsent, 586
 wait for a card to be inserted or removed —
 SCF_Terminal_waitForCardPresent, 586
 wait for PICL tree to refresh — picl_wait, 450
 walk objects within resource pool
 configurations —
 pool_walk_components, 482
 walk objects within resource pool
 configurations — pool_walk_pools, 482

walk objects within resource pool
 configurations — pool_walk_resources, 482

walk subtree by class —
 picl_walk_tree_by_class, 451

walk subtree by class —
 ptree_walk_tree_by_class, 514

write to a segment —
 rsm_memseg_import_put16, 548

write to a segment —
 rsm_memseg_import_put32, 548

write to a segment —
 rsm_memseg_import_put64, 548

write to a segment —
 rsm_memseg_import_put8, 548

write to a segment —
 rsm_memseg_import_put, 548

write to a segment using a list of I/O requests
 — rsm_memseg_import_getv, 550

write to a segment using a list of I/O requests
 — rsm_memseg_import_putv, 550

write_vtoc — read and write a disk's
 VTOC, 515

wsreg_add_child_component — add or remove
 a child component, 680

wsreg_add_compatible_version — add or
 remove a backward compatible version, 682

wsreg_add_dependent_component — add or
 remove a dependent component, 684

wsreg_add_display_name — add, remove, or
 return a localized display name, 686

wsreg_add_required_component — add or
 remove a required component, 688

wsreg_can_access_registry — determine access
 to product install registry, 690

wsreg_clone_component — clone a
 component, 692

wsreg_components_equal — determine equality
 of two components, 693

wsreg_create_component — create or release a
 component, 694

wsreg_free_component — create or release a
 component, 694

wsreg_free_component_array — create or
 release a component, 694

wsreg_get_child_components — add or remove
 a child component, 680

wsreg_get_compatible_versions — add or
 remove a backward compatible version, 682

wsreg_get_data — add or retrieve a key-value
 pair, 705

wsreg_get_data_pairs — add or retrieve a
 key-value pair, 705

wsreg_get_dependent_components — add or
 remove a dependent component, 684

wsreg_get_display_languages — add, remove,
 or return a localized display name, 686

wsreg_get_display_name — add, remove, or
 return a localized display name, 686

wsreg_get_id — set or get the uuid of a
 component, 707

wsreg_get_instance — set or get the instance of
 a component, 708

wsreg_get_location — set or get the location of
 a component, 710

wsreg_get_parent — set or get the parent of a
 component, 711

wsreg_get_required_components — add or
 remove a required component, 688

wsreg_get_type — set or get the type of a
 component, 712

wsreg_get_uninstaller — set or get the
 uninstaller of a component, 713

wsreg_get_unique_name — set or get the
 unique name of a component, 714

wsreg_get_vendor — set or get the vendor of a
 component, 715

wsreg_get_version — set or get the version of a
 component, 716

wsreg_initialize — initialize wsreg library, 696

wsreg_query_create — create a new query, 697

wsreg_query_free — create a new query, 697

wsreg_query_get_id — set or get the uuid of a
 query, 698

wsreg_query_get_instance — set or get the
 instance of a query, 699

wsreg_query_get_location — set or get the
 location of a query, 700

wsreg_query_get_unique_name — set or get the
 unique name of a query, 701

wsreg_query_get_version — set or get the
 version of a query, 702

wsreg_query_set_id — set or get the uuid of a
 query, 698

wsreg_query_set_instance — set or get the
 instance of a query, 699

`wsreg_query_set_location` — set or get the location of a query, 700
`wsreg_query_set_unique_name` — set or get the unique name of a query, 701
`wsreg_query_set_version` — set or get the version of a query, 702
`wsreg_register` — register a component in the product install registry, 703
`wsreg_remove_child_component` — add or remove a child component, 680
`wsreg_remove_compatible_version` — add or remove a backward compatible version, 682
`wsreg_remove_dependent_component` — add or remove a dependent component, 684
`wsreg_remove_display_name` — add, remove, or return a localized display name, 686
`wsreg_remove_required_component` — add or remove a required component, 688
`wsreg_set_data` — add or retrieve a key-value pair, 705
`wsreg_set_id` — set or get the uuid of a component, 707
`wsreg_set_instance` — set or get the instance of a component, 708
`wsreg_set_location` — set or get the location of a component, 710
`wsreg_set_parent` — set or get the parent of a component, 711
`wsreg_set_type` — set or get the type of a component, 712
`wsreg_set_uninstaller` — set or get the uninstaller of a component, 713
`wsreg_set_unique_name` — set or get the unique name of a component, 714
`wsreg_set_vendor` — set or get the vendor of a component, 715
`wsreg_set_version` — set or get the version of a component, 716
`wsreg_unregister` — remove a component from the product install registry, 717

Y

`y0` — Bessel functions of the second kind, 719
`y1` — Bessel functions of the second kind, 719
`yn` — Bessel functions of the second kind, 719