man pages section 3: Extended
Library Functions
## Contents

**Preface**  13

**Extended Library Functions**  19

- aclcheck(3SEC)  20
- aclsort(3SEC)  22
- acltomode(3SEC)  23
- acltotext(3SEC)  24
- acos(3M)  26
- acosh(3M)  27
- asin(3M)  28
- atan2(3M)  29
- atan(3M)  30
- au_open(3BSM)  31
- au_preselect(3BSM)  33
- au_to(3BSM)  35
- au_user_mask(3BSM)  39
- bgets(3GEN)  41
- bufsplit(3GEN)  43
- cbrt(3M)  44
- ceil(3M)  45
- config_admin(3CFGADM)  46
- ConnectToServer(3DMI)  54
- copylist(3GEN)  55
- copysign(3M)  56
- cos(3M)  57
- cosh(3M)  58
<table>
<thead>
<tr>
<th>Function</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getauusername</code></td>
<td>3BSM</td>
</tr>
<tr>
<td><code>getddent</code></td>
<td>3BSM</td>
</tr>
<tr>
<td><code>getdmapent</code></td>
<td>3BSM</td>
</tr>
<tr>
<td><code>getexecattr</code></td>
<td>3SECDB</td>
</tr>
<tr>
<td><code>getfauditflags</code></td>
<td>3BSM</td>
</tr>
<tr>
<td><code>getprofattr</code></td>
<td>3SECDB</td>
</tr>
<tr>
<td><code>getprojent</code></td>
<td>3PROJECT</td>
</tr>
<tr>
<td><code>getuserattr</code></td>
<td>3SECDB</td>
</tr>
<tr>
<td><code>gmatch</code></td>
<td>3GEN</td>
</tr>
<tr>
<td><code>hypot</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>ilogb</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>isencrypt</code></td>
<td>3GEN</td>
</tr>
<tr>
<td><code>isnan</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>j0</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>kstat</code></td>
<td>3EXT</td>
</tr>
<tr>
<td><code>kstat</code></td>
<td>3KSTAT</td>
</tr>
<tr>
<td><code>kstat_chain_update</code></td>
<td>3KSTAT</td>
</tr>
<tr>
<td><code>kstat_lookup</code></td>
<td>3KSTAT</td>
</tr>
<tr>
<td><code>kstat_open</code></td>
<td>3KSTAT</td>
</tr>
<tr>
<td><code>kstat_read</code></td>
<td>3KSTAT</td>
</tr>
<tr>
<td><code>kva_match</code></td>
<td>3SECDB</td>
</tr>
<tr>
<td><code>kvm_getu</code></td>
<td>3KVM</td>
</tr>
<tr>
<td><code>kvm_nextproc</code></td>
<td>3KVM</td>
</tr>
<tr>
<td><code>kvm_nlist</code></td>
<td>3KVM</td>
</tr>
<tr>
<td><code>kvm_open</code></td>
<td>3KVM</td>
</tr>
<tr>
<td><code>kvm_read</code></td>
<td>3KVM</td>
</tr>
<tr>
<td><code>ld_support</code></td>
<td>3EXT</td>
</tr>
<tr>
<td><code>lgamma</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>libdevinfo</code></td>
<td>3DEVINFO</td>
</tr>
<tr>
<td><code>libnvpair</code></td>
<td>3NVPAIR</td>
</tr>
<tr>
<td><code>libpicl</code></td>
<td>3PICL</td>
</tr>
<tr>
<td><code>libpicltree</code></td>
<td>3PICLTREE</td>
</tr>
<tr>
<td><code>libtnfctl</code></td>
<td>3TNF</td>
</tr>
<tr>
<td><code>log10</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>log1p</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>log</code></td>
<td>3M</td>
</tr>
<tr>
<td><code>logb</code></td>
<td>3M</td>
</tr>
</tbody>
</table>

6  man pages section 3: Extended Library Functions • May 2002
matherr(3M) 290
m_create_layout(3LAYOUT) 296
md5(3EXT) 298
m_destroy_layout(3LAYOUT) 300
media_findname(3VOLMGT) 301
media_getattr(3VOLMGT) 304
media_getid(3VOLMGT) 306
m_getvalues_layout(3LAYOUT) 307
mkdirp(3GEN) 308
mp(3MP) 310
m_setvalues_layout(3LAYOUT) 312
m_transform_layout(3LAYOUT) 313
m_wtransform_layout(3LAYOUT) 318
newDmiOctetString(3DMI) 324
newDmiString(3DMI) 325
nextafter(3M) 326
nlist(3ELF) 327
NOTE(3EXT) 328
nvlist_add_boolean(3NVPAIR) 330
nvlist_alloc(3NVPAIR) 332
nvlist_lookup_boolean(3NVPAIR) 337
nvlist_next_nvpair(3NVPAIR) 339
nvlist_remove(3NVPAIR) 341
nvpair_value_byte(3NVPAIR) 342
p2open(3GEN) 344
pam(3PAM) 346
pam_acct_mgmt(3PAM) 349
pamAuthenticate(3PAM) 350
pam_chauthtok(3PAM) 352
pam_getenv(3PAM) 354
pam_getenvlist(3PAM) 355
pam_get_user(3PAM) 356
pam_open_session(3PAM) 358
pam_putenv(3PAM) 360
pam_setcred(3PAM) 362
pam_set_data(3PAM) 364
pam_set_item(3PAM) 366
pam_sm(3PAM) 368
pam_sm_acct_mgmt(3PAM) 372
pam_sm_authenticate(3PAM) 374
pam_sm_chauthtok(3PAM) 376
pam_sm_open_session(3PAM) 379
pam_sm_setcred(3PAM) 381
pam_start(3PAM) 383
pam_strerror(3PAM) 386
pathfind(3GEN) 387
pctx_capture(3CPC) 389
pctx_set_events(3CPC) 391
picld_log(3PICLTREE) 394
picld_plugin_register(3PICLTREE) 395
picl_get_first_prop(3PICL) 397
picl_get_next_by_row(3PICL) 399
picl_get_prop_by_name(3PICL) 400
picl_get_propinfo(3PICL) 401
picl_get_propinfo_by_name(3PICL) 402
picl_get_propval(3PICL) 403
picl_get_root(3PICL) 405
picl_initialize(3PICL) 406
picl_set_propval(3PICL) 407
picl_shutdown(3PICL) 409
picl_strerror(3PICL) 410
picl_wait(3PICL) 411
picl_walk_tree_by_class(3PICL) 412
pool_associate(3POOL) 413
pool_component_info(3POOL) 416
pool_component_to_elem(3POOL) 418
pool_conf_alloc(3POOL) 419
pool_dynamic_location(3POOL) 424
pool_error(3POOL) 426
pool_get_binding(3POOL) 428
pool_get_pool(3POOL) 431
pool_get_property(3POOL) 433
pool_resource_create(3POOL) 436
pool_value_alloc(3POOL) 440
pool_walk_components(3POOL) 443
pow(3M) 445
printDmiAttributeValues(3DMI) 446
printDmiDataUnion(3DMI) 447
printDmiString(3DMI) 448
project(3EXT) 449
project_walk(3PROJECT) 451
ptree_add_node(3PICLTREE) 453
ptree_add_prop(3PICLTREE) 454
ptree_create_and_add_node(3PICLTREE) 455
ptree_create_and_add_prop(3PICLTREE) 456
ptree_create_node(3PICLTREE) 457
ptree_create_prop(3PICLTREE) 458
ptree_create_table(3PICLTREE) 460
ptree_find_node(3PICLTREE) 461
ptree_get_first_prop(3PICLTREE) 462
ptree_get_next_by_row(3PICLTREE) 463
ptree_get_node_by_path(3PICLTREE) 464
ptree_get_prop_by_name(3PICLTREE) 466
ptree_get_propinfo(3PICLTREE) 467
ptree_get_propinfo_by_name(3PICLTREE) 468
ptree_get_propval(3PICLTREE) 469
ptree_get_root(3PICLTREE) 470
ptree_init_propinfo(3PICLTREE) 471
ptree_post_event(3PICLTREE) 472
ptree_register_handler(3PICLTREE) 473
ptree_unregister_handler(3PICLTREE) 474
ptree_update_propval(3PICLTREE) 475
ptree_walk_tree_by_class(3PICLTREE) 476
read_vtoc(3EXT) 477
reg_ci_callback(3DMI) 478
regexpr(3GEN) 479
remainder(3M) 482
rint(3M) 483
rsm_create_localmemory_handle(3RSM) 484
rsm_get_controller(3RSM) 486
rsm_get_interconnect_topology(3RSM) 488
rsm_get_segmentid_range(3RSM) 490
rsm_intr_signal_post(3RSM) 492
rsm_memseg_export_create(3RSM) 494
rsm_memseg_export_publish(3RSM) 497
rsm_memseg_get_pollfd(3RSM) 500
rsm_memseg_import_connect(3RSM) 501
rsm_memseg_import_get(3RSM) 503
rsm_memseg_import_init_barrier(3RSM) 505
rsm_memseg_import_map(3RSM) 506
rsm_memseg_import_open_barrier(3RSM) 508
rsm_memseg_import_put(3RSM) 510
rsm_memseg_import_putv(3RSM) 512
rsm_memseg_import_set_mode(3RSM) 514
rtld_audit(3EXT) 515
rtld_db(3EXT) 516
scalb(3M) 517
scalbn(3M) 518
sendfile(3EXT) 519
sendfilev(3EXT) 522
setproject(3PROJECT) 525
significand(3M) 527
sin(3M) 528
sinh(3M) 529
sqrt(3M) 530
SSAAgentIsAlive(3SNMP) 531
SSAObjectId(3SNMP) 534
SSAStringCpy(3SNMP) 536
strccpy(3GEN) 537
strfind(3GEN) 539
sysevent_bind_handle(3SYSEVENT) 540
sysevent_free(3SYSEVENT) 542
sysevent_get_attr_list(3SYSEVENT) 543
sysevent_get_class_name(3SYSEVENT) 544
sysevent_get_vendor_name(3SYSEVENT) 546
sysevent_post_event(3SYSEVENT) 548
sysevent_subscribe_event(3SYSEVENT) 550
tan(3M) 553
tanh(3M) 554
tnfctl_buffer_alloc(3TNF) 555	nfctl_close(3TNF) 557	nfctl_indirect_open(3TNF) 559
wsreg_query_set_version(3WSREG) 632
wsreg_register(3WSREG) 633
wsreg_set_data(3WSREG) 635
wsreg_set_id(3WSREG) 637
wsreg_set_instance(3WSREG) 638
wsreg_set_location(3WSREG) 640
wsreg_set_parent(3WSREG) 641
wsreg_set_type(3WSREG) 642
wsreg_set_uninstaller(3WSREG) 643
wsreg_set_unique_name(3WSREG) 644
wsreg_set_vendor(3WSREG) 645
wsreg_set_version(3WSREG) 646
wsreg_unregister(3WSREG) 647
y0(3M) 649

Index 651
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:

[ ] 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 ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mtcio(7I).

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

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

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

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

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

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

Commands
Modifiers
Variables
Expressions
Input Grammar

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

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

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

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

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

SEE ALSO
This section lists references to other man pages, in-house documentation, and outside publications.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAGNOSTICS</td>
<td>This section lists diagnostic messages with a brief explanation of the condition causing the error.</td>
</tr>
<tr>
<td>WARNINGS</td>
<td>This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.</td>
</tr>
<tr>
<td>NOTES</td>
<td>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.</td>
</tr>
<tr>
<td>BUGS</td>
<td>This section describes known bugs and, wherever possible, suggests workarounds.</td>
</tr>
</tbody>
</table>
Extended Library Functions
aclcheck(3SEC)

NAME  aclcheck – check the validity of an ACL

SYNOPSIS  
```c
#include <sys/acl.h>

int aclcheck(aclent_t *aclbufp, int nentries, int *which);
```

DESCRIPTION  
The `aclcheck()` function checks the validity of an ACL pointed to by `aclbufp`. The `nentries` argument is the number of entries contained in the buffer. The `which` parameter returns the index of the first entry that is invalid.

The function verifies that an ACL pointed to by `aclbufp` is valid according to the following rules:

- There must be exactly one `GROUP_OBJ` ACL entry.
- There must be exactly one `USER_OBJ` ACL entry.
- There must be exactly one `OTHER_OBJ` ACL entry.
- If there are any `GROUP` ACL entries, then the group ID in each group ACL entry must be unique.
- If there are any `USER` ACL entries, then the user ID in each user ACL entry must be unique.
- If there are any `GROUP` or `USER` ACL entries, then there must be exactly one `CLASS_OBJ` (ACL mask) entry.
- If there are any default ACL entries, then the following apply:
  - There must be exactly one default `GROUP_OBJ` ACL entry.
  - There must be exactly one default `OTHER_OBJ` ACL entry.
  - There must be exactly one default `USER_OBJ` ACL entry.
  - If there are any `DEF_GROUP` entries, then the group ID in each `DEF_GROUP` ACL entry must be unique.
  - If there are any `DEF_USER` entries, then the user ID in each `DEF_USER` ACL entry must be unique.
  - If there are any `DEF_GROUP` or `DEF_USER` entries, then there must be exactly one `DEF_CLASS_OBJ` (default ACL mask) entry.
- If any of the above rules are violated, then the function fails with `errno` set to `EINVAL`.

RETURN VALUES  
If the ACL is valid, `aclcheck()` will return 0. Otherwise `errno` is set to `EINVAL` and return code is set to one of the following:

- `GRP_ERROR`: There is more than one `GROUP_OBJ` or `DEF_GROUP_OBJ` ACL entry.
- `USER_ERROR`: There is more than one `USER_OBJ` or `DEF_USER_OBJ` ACL entry.
- `CLASS_ERROR`: There is more than one `CLASS_OBJ` (ACL mask) or `DEF_CLASS_OBJ` (default ACL mask) entry.
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 which parameter returns −1 in this case.

MEM_ERROR The system cannot allocate any memory. The which parameter returns −1 in this case.

**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>Unsafe</td>
</tr>
</tbody>
</table>

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

NAME aclsort – sort an ACL

SYNOPSIS

#include <sys/acl.h>

int aclsort(int nentries, int calclass, aclent_t *aclbufp);

DESCRIPTION

The aclbufp argument points to a buffer containing ACL entries. The nentries argument specifies the number of ACL entries in the buffer. The calclass 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 aclsort() 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 aclsort() 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 function returns 0. Otherwise, it returns -1.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

acl(2), aclcheck(3SEC), attributes(5)
NAME
acltomode, aclfrommode — convert an ACL to or from permission bits

SYNOPSIS
cce [ 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);

DESCRIPTION
The acltomode() function converts an ACL pointed to by aclbufp into the permission bits buffer pointed to by modep. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, then the function fails with errno set to EINVAL.

The USER_OBJ ACL entry permission bits are copied to the file owner class bits in the permission bits buffer. The OTHER_OBJ ACL entry permission bits are copied to the file other class bits in the permission bits buffer. If there is a CLASS_OBJ (ACL mask) entry, the CLASS_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer. Otherwise, the GROUP_OBJ ACL entry permission bits are copied to the file group class bits in the permission bits buffer.

The aclfrommode() function converts the permission bits pointed to by modep into an ACL pointed to by aclbufp. If the USER_OBJ ACL entry, GROUP_OBJ ACL entry, or the OTHER_OBJ ACL entry cannot be found in the ACL buffer, the function fails with errno set to EINVAL.

The file owner class bits from the permission bits buffer are copied to the USER_OBJ ACL entry. The file other class bits from the permission bits buffer are copied to the OTHER_OBJ ACL entry. If there is a CLASS_OBJ (ACL mask) entry, the file group class bits from the permission bits buffer are copied to the CLASS_OBJ ACL entry, and the GROUP_OBJ ACL entry is not modified. Otherwise, the file group class bits from the permission bits buffer are copied to the GROUP_OBJ ACL entry.

The nentries argument represents the number of ACL entries in the buffer pointed to by aclbufp.

RETURN VALUES
Upon successful completion, the function returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
acl(2), attributes(5)
The acltotext() function converts an internal ACL representation pointed to by aclbufp into an external ACL representation. The space for the external text string is obtained using malloc(3C). The caller is responsible for freeing the space upon completion.

The aclfromtext() function converts an external ACL representation pointed to by acltextp into an internal ACL representation. The space for the list of ACL entries is obtained using malloc(3C). The caller is responsible for freeing the space upon completion. The aclcnt argument indicates the number of ACL entries found.

An external ACL representation is defined as follows:

```
<acl_entry>[,<acl_entry>]
```

Each <acl_entry> 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:

- **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 object.
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:

<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>Unsafe</td>
</tr>
</tbody>
</table>

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

NAME
acos – arc cosine function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double acos(double x);

DESCRIPTION
The acos() 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, acos() 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 ±Inf or NaN, either 0.0 or
NaN is returned and errno is set to EDOM.

If x is NaN, NaN is returned. If x is ±Inf, either 0.0 is returned and errno is set to
EDOM, or NaN is returned and errno may be set to EDOM.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by
Standards other than XPG4.

ERRORS
The acos() function will fail if:
    EDOM   The value x is not ±Inf or NaN and is not in the range [-1,1].

The acos() function may fail if:
    EDOM   The value x is ±Inf.

USAGE
An application wishing to check for error situations should set errno to 0 before
calling acos(). If errno is non-zero on return, or the value NaN is returned, 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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
cos(3M), isnan(3M), matherr(3M), attributes(5), standards(5)
acosh, asinh, atanh – inverse hyperbolic functions

SYNOPSIS

cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double acosh(double x);
double asinh(double x);
double atanh(double x);

DESCRIPTION

The acosh(), asinh() and atanh() functions compute the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.

RETURN VALUES

The acosh(), asinh() and atanh() functions return the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.

The acosh() function returns NaN and sets errno to EDOM when its argument is less than 1.0.

The atanh() function returns NaN and sets errno to EDOM when its argument has absolute value greater than 1.0.

The atanh() function returns ±Inf and sets errno to ERANGE when its argument is ±1.0.

If x is NaN, the asinh(), acosh() and atanh() functions return NaN.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS

The acosh() function will fail if:

EDOM The x argument is less than 1.0.

The atanh() 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 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>

SEE ALSO
cosh(3M), matherr(3M), sinh(3M), tanh(3M), attributes(5), standards(5)
The `asin()` function computes the principal value of the arc sine of `x`. The value of `x` should be in the range \([-1,1]\).

Upon successful completion, `asin()` 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 ±Inf or NaN, either 0.0 or NaN is returned and `errno` is set to `EDOM`.

If `x` is NaN, NaN is returned.

If `x` is ±Inf, either 0.0 is returned and `errno` is set to `EDOM` or NaN is returned and `errno` may be set to `EDOM`.

For exceptional cases, `matherr(3M)` tabulates the values to be returned as dictated by standards other than XPG4.

The `asin()` function will fail if:

- **EDOM**: The value `x` is not ±Inf or NaN and is not in the range \([-1,1]\).

The `asin()` function may fail if:

- **EDOM**: The value of `x` is ±Inf.

An application wishing to check for error situations should set `errno` to 0, then call `asin()`. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

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>

See also `isnan(3M), matherr(3M), sin(3M), attributes(5), standards(5)`
NAME
atan2 – arc tangent function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double atan2(double y, double x);

DESCRIPTION
The atan2() 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, atan2() returns the arc tangent of y/x in the range
[-pi,pi] radians. If both arguments are 0.0, 0.0 is returned and errno may be set to
EDOM.

If x or y is NaN, NaN is returned.

In IEEE 754 mode atan2() handles the following exceptional arguments in the spirit

atan2(±0, x) returns ±0 for x > 0 or x = +0;
atan2(±0, x) returns ±pi for x < 0 or x = −0;
atan2(y, ±0) returns pi/2 for y > 0;
atan2(y, ±0) returns −pi/2 for y < 0;
atan2(±y, Inf) returns ±0 for finite y > 0;
atan2(±Inf, x) returns ±pi/2 for finite x;
atan2(±y, −Inf) returns ±pi for finite y > 0;
atan2(±Inf, Inf) returns ±pi/4;
atan2(±Inf, −Inf) returns ±3pi/4.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by
Standards other than XPG4.

ERRORS
The atan2() function may fail if:

EDOM Both arguments are 0.0.

USAGE
An application wishing to check for error situations should set errno to 0 before
calling atan2(). 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>
<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>

SEE ALSO
atan(3M), isnan(3M), matherr(3M), tan(3M), attributes(5), standards(5)
atan(3M)

NAME  atan – arc tangent function

SYNOPSIS  cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double atan(double x);

DESCRIPTION  The atan() function computes the principal value of the arc tangent of x.

RETURN VALUES  Upon successful completion, atan() returns the arc tangent of x in the range
[-pi/2,pi/2] radians.

If x is NaN, NaN is returned.

If x is ±Inf, ±pi/2 is returned.

ERRORS  No errors will occur.

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>

SEE ALSO  atan2(3M), isnan(3M), tan(3M), attributes(5)
au_open, au_close, au_write – construct and write audit records

SYNOPSIS

cc [ flag ... ] file ... -lbserv -lsocket -lns1 -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);

DESCRIPTION

The au_open() function returns an audit record descriptor to which audit tokens can be written using au_write(). The audit record descriptor is an integer value that identifies a storage area where audit records are accumulated.

The au_close() function terminates the life of an audit record d of type event started by au_open(). If the keep parameter is AU_TO_NO_WRITE, the data contained therein is discarded. If the keep parameter is AU_TO_WRITE, the additional parameters are used to create a header token. Depending on the audit policy information obtained by auditon(2), additional tokens such as sequence and trailer tokens can be added to the record. The au_close() function then writes the record to the audit trail by calling audit(2). Any memory used is freed by calling free(3C).

The au_write() function adds the audit token pointed to by m to the audit record identified by the descriptor d. After this call is made the audit token is no longer available to the caller.

RETURN VALUES

Upon successful completion, au_open() returns an audit record descriptor. If a descriptor could not be allocated, au_open() returns −1 and sets errno to indicate the error.

Upon successful completion, au_close() returns 0. If d is an invalid or corrupted descriptor or if audit() fails, au_close() returns −1 without setting errno. If audit() fails, errno is set to one of the error values described on the audit(2) manual page.

Upon successful completion, au_write() returns 0. If d is an invalid descriptor or m is an invalid token, or if audit() fails, au_write() returns −1 without setting errno. If audit() fails, errno is set to one of the error values described on the audit(2) manual page.

ERRORS

The au_open() function will fail if:

-ENOMEM The physical limits of the system have been exceeded such that sufficient memory cannot be allocated.
-ENOMEM The physical limits of the system have been exceeded such that sufficient memory cannot be allocated.
-ENORANGE There is currently insufficient memory available. The application can try again later.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
au_open(3BSM)

<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>

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.
au_preselect(3BSM)

NAME 
au_preselect – preselect an audit event

SYNOPSIS
ce [ flag ... ] file... -lbsm -lsocket -lsm -lintl [ library ... ]
#include <bsm/libbsm.h>

int au_preselect(au_event_t event, au_mask_t *mask_p, int sorf, int flag);

DESCRIPTION
au_preselect() determines whether or not the audit event event is preselected against the binary preselection mask pointed to by mask_p (usually obtained by a call to getaudit(2)). au_preselect() looks up the classes associated with event in audit_event(4) and compares them with the classes in mask_p. If the classes associated with event match the classes in the specified portions of the binary preselection mask pointed to by mask_p, the event is said to be preselected.

sorf indicates whether the comparison is made with the success portion, the failure portion or both portions of the mask pointed to by mask_p.

The following are the valid values of sorf:

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.

flag 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 flag determines where au_preselect() obtains audit event information. The following are the valid values of flag:

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:

0 event is not preselected.

1 event is preselected.

−1 An error occurred. au_preselect() couldn’t allocate memory or couldn’t find event in the audit_event(4) database.

FILES
/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:

<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>

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.
au_to(3BSM)

**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_iport, au_to_me, au_to_newgroups,
au_toOpaque, au_to_path, au_to_process, au_to_process_ex, au_to_return,
auto_toReturn32, au_to_return64, au_to_socket, au_to_subject, au_to_subject_ex,
auto_to_text – create audit record tokens

**SYNOPSIS**
cc [ flag ... ] file ... -lsem -lssocket -lns1 -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_iport(u_short_t iport);
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(au_id_t auid, uid_t euid, gid_t egid, uid_t
                       ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_t *tid);
token_t *au_to_process_ex(au_id_t auid, uid_t euid, gid_t egid, uid_t
                        ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_addr_t *tid);
token_t *au_to_return(char number, uin32t_t value);
token_t *au_to_return32(char number, uin32t_t value);
token_t *au_to_return64(char number, uin64t_t value);
token_t *au_to_socket(struct oldsocket *so);
token_t *au_to_subject(au_id_t auid, uid_t euid, gid_t egid, uid_t
                      ruid, gid_t rgid, pid_t pid, au_asid_t sid, au_tid_t *tid);
au_to(3BSM)

DESCRIPTION

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 \( \text{unit}_{\text{print}} \) parameter determines the preferred display base of the data and
is one of \( \text{AUP}_\text{BINARY}, \text{AUP}_\text{OCTAL}, \text{AUP}_\text{DECIMAL}, \text{AUP}_\text{HEX}, \text{or AUP}_\text{STRING} \). The
\( \text{unit}_{\text{type}} \) parameter defines the basic unit of data and is one of \( \text{AUR}_\text{BYTE}, \text{AUR}_\text{CHAR},
\text{AUR}_\text{SHORT}, \text{AUR}_\text{INT}, \text{or AUR}_\text{LONG} \). The \( \text{unit}_{\text{count}} \) 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 \( internet_{\text{addr}} \) 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 `auid` (audit user ID), an `euid` (effective user ID), an `egid` (effective group ID), a `ruid` (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 (i.e. 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 `auid` (audit user ID), an `euid` (effective user ID), an `egid` (effective group ID), a `ruid` (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 formats the data pointed to by `so` into a “socket token.”

The `au_to_subject()` function formats an `auid` (audit user ID), an `euid` (effective user ID), an `egid` (effective group ID), a `ruid` (real user ID), a `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 `auid` (audit user ID), an `euid` (effective user ID), an `egid` (effective group ID), a `ruid` (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:

<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>

**SEE ALSO**

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

NAME
au_user_mask – get user’s binary preselection mask

SYNOPSIS
cc [ flag ... ] file ... -bssl -lsoket -lnsl -lntl [ library ... ]
#include <bsm/libbsm.h>

int au_user_mask(char *username, au_mask_t **mask_p);

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>

SEE ALSO
login(1), bsmconv(1M), getaudit(2), setaudit(2), au_preselect(3BSM), getacinfo(3BSM), getauusername(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.
bgets(3GEN)

NAME  bgets — read stream up to next delimiter

SYNOPSIS  cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *bgets (char *buffer, size_t count, FILE *stream, const char *breakstring);

DESCRIPTION  The bgets() function reads characters from stream into buffer until either count is exhausted or one of the characters in breakstring 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 breakstring 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

bgets(buffer, sizeof buffer, stream, "\n");

is identical to

fgets(buffer, sizeof buffer, stream);

There is always enough room reserved in the buffer for the trailing null character.

If breakstring is a null pointer, the value of breakstring from the previous call is used. If breakstring 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 bgets() function.

The following example prints the name of the first user encountered in /etc/passwd, including a trailing ":"?

#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;
}

Extended Library Functions  41
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>

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
cc [ flag ...] file ... -lgen [ library ...]
#include <libgen.h>

size_t bufsplit(char *buf, size_t n, char **a);

DESCRIPTION
bufsplit() examines the buffer, buf, and assigns values to the pointer array, a, so
that the pointers point to the first n fields in buf that are delimited by TABs or
NEWLINEs.

To change the characters used to separate fields, call bufsplit() with buf pointing
to the string of characters, and n and a set to zero. For example, to use colon (:), period
(,), and comma (,), as separators along with TAB and NEWLINE:

bufsplit (":.\t\n", 0, (char**)0);

RETURN VALUES
The number of fields assigned in the array a. If buf 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 bufsplit() function.

/*
 * set a[0] = "This", a[1] = "is", a[2] = "a",
 * a[3] = "test"
 */
bufsplit("This\tis\ta\tttest\n", 4, a);

NOTES
bufsplit() changes the delimiters to null bytes in buf.

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>
<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>

SEE ALSO
attributes(5)
cbrt(3M)

NAME cbrt – cube root function

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double cbrt(double x);

DESCRIPTION The cbrt() function computes the cube root of x.

RETURN VALUES On successful completion, cbrt() returns the cube root of x. If x is NaN, cbrt() returns NaN.

ERRORS No errors will occur.

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>

SEE ALSO attributes(5)
#include <math.h>

double ceil(double x);

The `ceil()` function computes the smallest integral value not less than `x`.

Upon successful completion, `ceil()` returns the smallest integral value not less than `x`, expressed as a type `double`.

If `x` is NaN, NaN is returned.

If `x` is ±Inf or ±0, `x` is returned.

No errors will occur.

The integral value returned by `ceil()` as a `double` may not be expressible as an `int` or `long int`. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

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>

`floor(3M)`, `isnan(3M)`, `attributes(5)`
The following interfaces have been deprecated and their use is strongly discouraged:

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct cfga_stat_data *buf, const char *options, char **errstring);
```
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 *listopts,
char **errstring, cfga_flags_t flags);

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 libcfgadm library is used to provide the services of the cfgadm(1M) command. The hardware-specific libraries are located in
/usr/platform/${machine}/lib/cfgadm,
/usr/platform/${arch}/lib/cfgadm, and /usr/lib/cfgadm. 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:
CFGA_CMD_INSERT, CFGA_CMD_REMOVE, CFGA_CMD_DISCONNECT,
CFGA_CMD_CONNECT, CFGA_CMD_CONFIGURE, or CFGA_CMD_UNCONFIGURE. The
state_change_cmd CFGA_CMD_INSERT is used to prepare for manual insertion or to
activate automatic hardware insertion of an occupant. The state_change_cmd
CFGA_CMD_REMOVE is used to prepare for manual removal or activate automatic
hardware removal of an occupant. The state_change_cmd CFGA_CMD_DISCONNECT is
used to disable normal communication to or from an occupant in a receptacle. The state_change_cmd CFGA_CMD_CONNECT is used to enable communication to or from an occupant in a receptacle. The state_change_cmd CFGA_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 CFGA_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.
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:

```c
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:

```c
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 `erstring`. If `erstring` is NULL no error message will be generated or returned. If `erstring` is not NULL and no error message is generated, the pointer referenced by `erstring` 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[CFGA_LOG_EXT_LEN];`
- `typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];`
- `typedef char cfga_class_t[CFGA_CLASS_LEN];`
- `typedef char cfga_info_t[CFGA_INFO_LEN];`
- `typedef char cfga_type_t[CFGA_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`. 
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-to-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:

```c
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 its localized help message.
The `config_*()` and `cfga_*()` functions return the following values. Additional error information may be returned through `errstring` if the return code is not `CFGA_OK`. See DESCRIPTION for details.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFGA_BUSY</td>
<td>The command was not completed due to an element of the system configuration administration system being busy.</td>
</tr>
<tr>
<td>CFGA_ATTR_INVAL</td>
<td>No attachment points with the specified attributes exist.</td>
</tr>
<tr>
<td>CFGA_ERROR</td>
<td>An error occurred during the processing of the requested operation. This error code includes validation of the command arguments by the hardware-specific code.</td>
</tr>
<tr>
<td>CFGA_INSUFFICIENT_CONDITION</td>
<td>Operation failed due to attachment point condition.</td>
</tr>
<tr>
<td>CFGA_INVAL</td>
<td>The system configuration administration operation requested is not supported on the specified attachment point.</td>
</tr>
<tr>
<td>CFGA_LIB_ERROR</td>
<td>A procedural error occurred in the library, including failure to obtain process resources such as memory and file descriptors.</td>
</tr>
<tr>
<td>CFGA_NACK</td>
<td>The command was not completed due to a negative acknowledgement from the <code>confp-&gt;confirm</code> function.</td>
</tr>
<tr>
<td>CFGA_NO_LIB</td>
<td>A hardware-specific library could not be located using the supplied <code>ap_id</code>.</td>
</tr>
<tr>
<td>CFGA_NOTSUPP</td>
<td>System configuration administration is not supported on the specified attachment point.</td>
</tr>
<tr>
<td>CFGA_OK</td>
<td>The command completed as requested.</td>
</tr>
<tr>
<td>CFGA_OPSUPS</td>
<td>System configuration administration operation is not supported on this attachment point.</td>
</tr>
<tr>
<td>CFGA_PRIV</td>
<td>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.</td>
</tr>
<tr>
<td>CFGA_SYSTEM_BUSY</td>
<td>The command required a service interruption and was not completed due to the required service interruption.</td>
</tr>
</tbody>
</table>

Extended Library Functions  51
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcsu, SUNWkvm</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**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)`
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`:

```c
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
cc [ flag ... ] file ... -ldmici -ldmimi [ library ... ]
#include <dmi/api.hh>

bool_t ConnectToServer(ConnectI *argp, DmiRpcHandle *dmi_rpc_handle);

DESCRIPTION
The ConnectToServer() function enables a management application or a
component instrumentation to connect to a DMI service provider.

The argp parameter is an input parameter that uses the following data structure:

```
struct ConnectIN {
    char *host;
    const char *nettype;
    ServerType servertype;
    RpcType rpctype;
}
```

The host member indicates the host on which the service provider is running. The
default is localhost.

The nettype member specifies the type of transport RPC uses. The default is netpath.

The servertype member indicates whether the connecting process is a management
application or a component instrumentation.

The rpctype member specifies the type of RPC, either ONC or DCE. Only ONC is
supported in the Solaris 7 release.

The dmi_rpc_handle parameter is the output parameter that returns DMI RPC handle.

RETURN VALUES
The ConnectToServer() function returns TRUE if successful, otherwise FALSE.

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>
</tbody>
</table>
```

SEE ALSO
DisconnectToServer(3DMI), attributes(5)
NAME

copylist – copy a file into memory

SYNOPSIS

cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *copylist(const char *filenn, off_t *szptr);

DESCRIPTION

The copylist() 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 filenn to the name of the file to be copied, and a pointer szptr to a variable where the size of the file will be stored.

Upon success, copylist() returns a pointer to the memory allocated. Otherwise it returns NULL if it has trouble finding the file, calling malloc(), or reading the file.

USAGE

The copylist() function has a transitional interface for 64-bit file offsets. See lf64(5).

EXAMPLES

EXAMPLE 1 Example of copylist() function.

/* 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);
}

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>

SEE ALSO

malloc(3C), attributes(5), lf64(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.
copysign(3M)

NAME    copysign – return magnitude of first argument and sign of second argument

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
    #include <math.h>
    double copysign(double x, double y);

DESCRIPTION The copysign() function returns a value with the magnitude of x and the sign of y.
It produces a NaN with the sign of y if x is a NaN.

RETURN VALUES The copysign() function returns a value with the magnitude of x and the sign of y.

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>

SEE ALSO attributes(5)
NAME  cos – cosine function

SYNOPSIS  
```
c [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double cos(double x);
```

DESCRIPTION  The \texttt{cos()} function computes the cosine of \( x \), measured in radians.

RETURN VALUES  Upon successful completion, \texttt{cos()} returns the cosine of \( x \).

If \( x \) is \texttt{NaN} or ±\texttt{Inf}, \texttt{NaN} is returned.

ERRORS  No errors will occur.

ATTRIBUTES  See \texttt{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>

SEE ALSO  \texttt{acos(3M)}, \texttt{isnan(3M)}, \texttt{sin(3M)}, \texttt{tan(3M)}, \texttt{attributes(5)}
The `cosh()` function computes the hyperbolic cosine of `x`.

Upon successful completion, `cosh()` returns the hyperbolic cosine of `x`.

If the result would cause an overflow, `HUGE_VAL` is returned and `errno` is set to `ERANGE`.

If `x` is NaN, NaN is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned as dictated by Standards other than XPG4.

The `cosh()` function will fail if:

- `ERANGE` The result would cause an overflow.

An application wishing to check for error situations should set `errno` to 0 before calling `cosh()`. If `errno` is non-zero on return, or the returned value is NaN, an error has occurred.

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>

See also `acosh(3M)`, `isnan(3M)`, `matherr(3M)`, `sinh(3M)`, `tanh(3M)`, attributes(5), standards(5)
The UltraSPARC and Pentium microprocessor families contain hardware performance counters 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.

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.

To fully support the two-level threads model in Solaris, it would be necessary to virtualize the performance counters to each thread. This version of the library does not allow per-thread data to be captured unless bound threads are used. Even without bound threads, however, the counters can still be used to assess aggregate program behavior.

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:

- `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
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 `cpc_access(3CPC)` function determines the accessibility of the counters and should be invoked before any attempt to program the counters.

Events are specified using a `getsubopt(3C)`-style language for both the events and the additional control bits that determine what causes the counters to increment. The `cpc_strtoevent()` function translates a string to an event specification which can then be used to program the counters. The `cpc_eventtostr()` function returns the canonical form of the string that corresponds to a particular event. The `cpc_getusage(3CPC)` function returns a string that specifies the syntax of the string, while `cpc_walk_names(3CPC)` 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 performance counter context 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, `libpctx`, provides a simple set of interfaces that use the facilities of `proc(4)` to control a target process, and together with functions in `libcpc`, allow `truss`-like tools to be constructed to measure the performance counters in other applications. An example of one such application is `cputrack(1)`.

The functions in `libpctx` are independent of those in `libcpc`. 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 `libpctx` and `libcpc` that consume `pctx_t` handles.
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).
Access to CPU performance counters is possible only on systems where the appropriate hardware exists and is correctly configured. The `cpc_access()` function must 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 `/proc`.

Upon successful completion, `cpc_access()` returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

By default, two common `errno` values are decoded and cause the library to print an error message using its reporting mechanism. See `cpc_seterrfn(3CPC)` for a description of how this behavior can be modified.

The `cpc_access()` function will fail if:

- **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.

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>
<tr>
<td>Availability</td>
<td>SUNWcpcu (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO** `cpc(3CPC), cpc_seterrfn(3CPC), proc(4), attributes(5)`
cpc_bind_event(3CPC)

NAME
cpc_bind_event, cpc_take_sample, cpc_rele – use CPU performance counters on lwps

SYNOPSIS
cc [ flag... ] file... -lcpc [ library... ]
#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);

DESCRIPTION
Once the events to be sampled have been selected using, for example, cpc_strtoevent(3CPC), the event selections can be bound to the calling LWP using cpc_bind_event(). If cpc_bind_event() 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 cpc_take_sample(), and dereferencing the fields of the ce_pic[] array returned. The ce_hrt field contains the timestamp at which the kernel last sampled the counters.

To immediately remove the performance counter context on an LWP, the cpc_rele() 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, cpc_bind_event() and cpc_take_sample() return 0. Otherwise, these functions return -1, and set errno to indicate the error.

ERRORS
The cpc_bind_event() and cpc_take_sample() functions will fail if:

EFAULT The event 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 cpc_bind_event(), this implies that no new contexts can be created. For cpc_take_sample(), this implies that the performance counter context has been invalidated and must be
cpc_bind_event(3CPC)

released with cpc_rele(). Robust programs should be coded to expect this behavior and recover from it by releasing the now invalid context by calling cpc_rele() sleeping for a while, then attempting to bind and sample the event once more.

EINVAL The cpc_take_sample() function has been invoked before the context is bound.

Prior to calling cpc_bind_event(), applications should call cpc_access(3CPC) 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 libcpc 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 PERFEVENTS environment variable to other strings (a list of which can be gleaned from the -h flag of the cpustat or cputrack utilities), other events can be counted. The error() routine below is assumed to be a user-provided routine analogous to the familiar printf(3C) routine from the C library but which also performs an exit(2) after printing the message.

```c
#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_strtorevent(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));
    return 0;
}
```

64 man pages section 3: Extended Library Functions • Last Revised 20 Mar 2001
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: %" PRI64d " %" PRI64d "\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
dm_handler(int sig, siginfo_t *sip, void *arg)
{
    uc_event_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 uc_context: %p 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)
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

```
rd %pic, %rN ! All UltraSPARC
wr %rN, %pic ! (ditto, but see text)
```

IA

```
rdpmc ! 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.
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.

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_setoevent(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 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 interrupts 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.
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.

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.

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>
<tr>
<td>Availability</td>
<td>SUNWcpcu (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO cputstat(1), cpc(3CPC), cpc_access(3CPC), cpc_strtoevent(3CPC), attributes(5)
cpc_count_usr_events(3CPC)

NAME
cpc_count_usr_events, cpc_count_sys_events – enable and disable performance counters

SYNOPSIS
cc [ flag ... ] file ... -lcpc [ library ... ]
#include <libcpc.h>

int cpc_count_usr_events(int enable);
int cpc_count_sys_events(int enable);

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 cpc_count_usr_events() function can be used to control whether events are counted on behalf of the application running in user mode, while cpc_count_sys_events() 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 enable argument is non-zero, counting of events is enabled, otherwise they are disabled.

RETURN VALUES
Upon successful completion, cpc_count_usr_events() and cpc_count_sys_events() return 0. Otherwise, the functions return −1 and set errno to indicate the error.

ERRORS
The cpc_count_usr_events() and cpc_count_sys_events() functions will fail if:

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 cpc_count_usr_events() to minimize code needed by application.

In this example, the routine cpc_count_usr_events() is used to minimize the amount of code that needs to be added to the application. The cputrack(1) 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 cputrack with the nouser 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.

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;

Extended Library Functions  69
cpc_count_usr_events(3CPC)

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

```c
/* ... 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:

<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**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)
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.

For UltraSPARC, the structure contains the following members:

```c
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;
```

For Pentium, the structure contains the following members:

```c
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;
```

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.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
</table>

Extended Library Functions 71
cpc_event(3CPC)

<table>
<thead>
<tr>
<th>Availability</th>
<th>SUNWcpcu</th>
</tr>
</thead>
</table>

**SEE ALSO**

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

NAME
cpc_event_diff, cpc_event_accum – simple difference and accumulate operations

SYNOPSIS
cc [ flag... ] file... -lcpc [ library... ]
#include <libcpc.h>

#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);

DESCRIPTION
The cpc_event_accum() and cpc_event_diff() functions perform common
accumulate and difference operations on cpc_event(3CPC) data structures. Use of
these functions increases program portability, since structure members are not
referenced directly.

cpc_event_accum() The cpc_event_accum() function adds the ce_pic fields of event into the
corresponding fields of accum. The ce_hrt field of accum is set to the later of the times
in event and accum.

SPARC:
The function adds the contents of the ce_tick field of event into the corresponding
field of accum.

IA:
The function adds the contents of the ce_tsc field of event into the corresponding
field of accum.

cpc_event_diff() The cpc_event_diff() function places the difference between the ce_pic fields of
after and before and places them in the corresponding field of diff. The ce_hrt field of
diff is set to the ce_hrt field of after.

SPARC:
Additionally, the function computes the difference between the ce_tick fields of after
and before, and places it in the corresponding field of diff.

IA:
Additionally, the function computes the difference between the ce_tsc fields of after
and before, and places it in the corresponding field of diff.

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)|
+----------------+-------------------+
cpc_event_diff(3CPC)

<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>

SEE ALSO  cpc(3CPC), cpc_event(3CPC), attributes(5).
The `cpc_getcpuver()` 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 `cpc_getcpuver()` function returns −1 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 `cpc_getcciname()` function returns a printable description of the processor
performance counter interfaces for example, the string `UltraSPARC I&II`. 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 `cpuver` argument is unrecognized, the
function returns NULL.

The `cpc_getcpuref()` 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 `cpuver` argument is
unrecognized, the function returns NULL.

The `cpc_getusage()` function returns a compact description of the
getsubopt()-oriented syntax that is consumed by `cpc_strtoevent(3CPC)`. It is
returned as a space-separated set of tokens to allow the caller to wrap lines at
convenient boundaries. If the `cpuver` argument is unrecognized, the function returns
NULL.

The `cpc_getnpic()` function returns the number of valid fields in the `ce_pic[]`
array of a `cpc_event_t` data structure.
cpc_getcpuver(3CPC)

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:

<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

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

NOTES
Only SPARC processors are described by the SPARC version of the library, and only Intel processors are described by the Intel version of the library.
cpc_pctx_bind_event(3CPC)

NAME

cpc_pctx_bind_event, cpc_pctx_take_sample, cpc_pctx_rele, cpc_pctx_invalidate –
access CPU performance counters in other processes

SYNOPSIS

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);

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
controlling process, to manipulate the performance counters in the context of a controlled
process. The controlled process is described by the pctx argument, which must be
obtained from an invocation of pctx_capture(3CPC) or pctx_create(3CPC) and
passed to the functions described on this page in the context of an event handler.

The semantics of the functions cpc_pctx_bind_event(),
cpc_pctx_take_sample(), and cpc_pctx_rele() are directly analogous to those
of cpc_bind_event(), cpc_take_sample(), and cpc_rele() described on the
cpc_bind_event(3CPC) manual page.

The cpc_pctx_invalidate() 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 errno to
indicate the error.

ERRORS

The cpc_pctx_bind_event(), cpc_pctx_take_sample(), and
cpc_pctx_rele() functions return the same errno values the analogous functions
described on the cpc_bind_event(3CPC) manual page. In addition, these function
may fail if:

ESRCH The value of the lwpid argument is invalid in the context of the
controlled process.

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>Unsafe</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcpcu (32-bit)</td>
</tr>
</tbody>
</table>
cpc_pctx_bind_event(3CPC)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**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.
cpc_seterrfn(3CPC)

NAME

cpc_seterrfn – control libcpc error reporting

SYNOPSIS

cc [ flag... ] file... -lcpc [ library... ] #include <libcpc.h>

typedef void(cpc_errfn_t)(const char *fn, const char *fmt, va_list ap);

void cpc_seterrfn(cpc_errfn_t *errfn);

DESCRIPTION

For the convenience of programmers instrumenting their code, several libcpc functions automatically emit to stderr 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 cpc_seterrfn() function allows the caller to provide an alternate function for reporting errors; the type signature is shown above. The fn argument is passed the library function name that detected the error, the format string fmt and argument pointer ap can be passed directly to vsnprintf(3C) or similar varargs-based routine for formatting.

The default printing routine can be restored by calling the routine with an errfn argument of NULL.

EXAMPLES

EXAMPLE 1 Debugging example.

This example produces error messages only when debugging the program containing it, or when the cpc_strtoevent() function is reporting an error when parsing an event specification

```c
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);
}
```

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>Unsafe</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcpcu (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>
cpc_seterrfn(3CPC)

SEE ALSO cpc(3CPC), vsnprintf(3C), attributes(5).
The `cpc_shared_open()` 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 `cpc_shared_bind_event()`, `cpc_shared_take_sample()`, and `cpc_shared_rele()` functions are directly analogous to the corresponding `cpc_bind_event()`, `cpc_take_sample()`, and `cpc_rele()` functions described on the `cpc_bind_event(3CPC)` manual page, except that they operate on the counters of a particular processor.

If a thread wishes to access the counters using this interface, it must do so using a thread bound to an lwp, (see the `THR_BOUND` flag to `thr_create(3THR)`), that has in turn bound itself to a processor using `processor_bind(2)`.

Unlike the `cpc_bind_event(3CPC)` 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 `cpc_shared_open()` will immediately invalidate all existing performance counter context on the system, and prevent all subsequent attempts to bind counter context to lwps from succeeding anywhere on the system until the last caller invokes `cpc_shared_close()`.

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 `cpc` 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 `CPC_BIND_LWP_INHERIT` and `CPC_BIND_EMT_OVF` flags are invalid for the shared interface.

On success, the functions (apart from `cpc_shared_close()`) return 0. On failure, the functions return −1 and set `errno`, 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 event 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:

<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO

processor_bind(2), cpc(3CPC), cpc_bind_event(3CPC), thr_create(3THR), attributes(5)
**NAME**
cpc_strtoevent, cpc_eventtostr – translate strings to and from events

**SYNOPSIS**
```
c [ flag... ] file... -lpc [ library... ]
#include <libcpc.h>

int cpc_strtoevent(int cpuver, const char *spec, cpc_event_t *event);
char *cpc_eventtostr(cpc_event_t *event);
```

**DESCRIPTION**
The `cpc_strtoevent()` function translates an event specification to the appropriate collection of control bits in a `cpc_event_t` structure pointed to by the `event` 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 `ce_cpuver` field and the ISA-dependent control registers of event are initialized appropriately, and the rest of the `cpc_event_t` structure is initialized to 0.

The `cpc_eventtostr()` function takes an event and constructs a compact canonical string representation for that event.

**RETURN VALUES**
Upon successful completion, `cpc_strtoevent()` 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 `cpc_seterrfn(3CPC)`).

Upon successful completion, `cpc_eventtostr()` returns a pointer to a string. The string returned must be freed by the caller using `free(3C)`. If `cpc_eventtostr()` 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 `cpc_getusage()` function (see `cpc_getcpuver(3CPC)`), but is explained in further detail below.

**UltraSPARC**
On UltraSPARC processors, the syntax for setting options is as follows:
```
pic0=<eventspec>,pic1=<eventspec> [,sys] [,nouser]
```
This syntax, which reflects the simplicity of the options available using the `%pcr` register, forces both counter events to be selected. By default only user events are counted; however, the `sys` keyword allows system (kernel) events to be counted as well. User event counting can be disabled by specifying the `nouser` keyword.

The keywords `pic0` and `pic1` 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:
```
pic0=<eventspec>,pic1=<eventspec> [,sys[0|1]] [,nouser[0|1]]
[noedge[0|1]] [,pc[0|1]]
```
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.

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

```
pic0=<eventspec>,pic1=<eventspec> [,sys[0|1]]
[,noedge[0|1]] [,nouser[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.

### Pentium II

```c
#include <stdio.h>

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);
```

### SPARC

**EXAMPLE 1** SPARC Example.

```c
#include <stdio.h>

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:

<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

### SEE ALSO

cpc(3CPC), cpc_getcpuver(3CPC), cpc_seterrfn(3CPC), free(3C), getsubopt(3C), attributes(5)
These functions are provided as a convenience only. As new processors are usually released asynchronously with software, the library allows the `pic0` and `pic1` keywords to interpret numeric values specified directly in hexadecimal, octal, or decimal.
cpc_version(3CPC)

NAME
cpc_version – coordinate CPC library and application versions

SYNOPSIS
cc [ flag... ] file... -lcpc [ library... ]
#include <libcpc.h>

uint_t cpc_version(uint_t version);

DESCRIPTION
The cpc_version() function takes an interface version as an argument and returns
an interface version as a result. Usually, the argument will be the value of
CPC_VER_CURRENT bound to the application when it was compiled.

RETURN VALUES
If the version requested is still supported by the implementation, cpc_version()
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, cpc_version() returns CPC_VER_NONE, indicating that the
application will at least need to be recompiled to operate correctly on the new
platform, and may require further changes.

If version is CPC_VER_NONE, cpc_version() 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:

if (cpc_version(CPC_VER_CURRENT) == CPC_VER_NONE) {
    /* version mismatch - library cannot translate */
    exit(1);
}

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>Unsafe</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcpcu (32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO
cpc(3CPC), attributes(5)

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 cpc_event_t
data structure.
NAME
demangle, cplus_demangle – decode a C++ encoded symbol name

SYNOPSIS
cce [ flag ... ] file [ library ... ] -ldemangle

#include <demangle.h>

int cplus_demangle(const char *symbol, char *prototype, size_t size);

DESCRIPTION
The cplus_demangle() 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 symbol string argument points to the input mangled name.

The prototype argument points to a user-specified output string buffer, of size bytes.

The cplus_demangle() function operates on mangled names generated by SPARCompilers C++ 3.0.1, 4.0.1, 4.1 and 4.2.

The cplus_demangle() function improves and replaces the demangle() function.

Refer to the cc.1, dem.1, and c++filt.1 manual pages in the /opt/SUNWspro/man/man1 directory. These pages are only available with the SPROcc package.

RETURN VALUES
The cplus_demangle() function returns the following values:

0 The symbol argument is a valid mangled name and prototype contains a (partial) prototype for the symbol.

DEMANGLE_ENAME The symbol argument is not a valid mangled name and the content of prototype is a copy of the symbol.

DEMANGLE_ESPACE The prototype output buffer is too small to contain the prototype (or the symbol), and the content of prototype is undefined.
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 `devid_get()` function returns in `retdevid` the device ID for the device associated with the open file descriptor `fd`, 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 `retdevid` using the `devid_free()` function.

The `devid_free()` function frees the space that was allocated for the returned `devid` by `devid_get()` and `devid_str_decode()`.

The `devid_get_minor_name()` function returns the minor name, in `retminor_name`, for the device associated with the open file descriptor `fd`. 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 `retminor_name` string using `devid_str_free()`.

The `devid_deviceid_to_nmlist()` function returns an array of `devid_nmlist` structures, where each entry matches the `devid` and `minor_name` passed in. If the `minor_name` specified is one of the special values (DEVID_MINOR_NAME_ALL, DEVID_MINOR_NAME_ALL_BLK, or DEVID_MINOR_NAME_ALL_BLK), then all minor names associated with `devid` which also meet the special `minor_name` filtering...
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 `retnname` 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()`.
The `devid_str_free()` function frees the character string returned by `devid_str_encode()` and the `retminor_name` argument returned by `devid_str_decode()`.

**RETURN VALUES**

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`.

```c
int fd;

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

<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>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

SEE ALSO
free(3C), libdevid(3LIB), attributes(5), ddi_devid_compare(9F)
di_binding_name(3DEVINFO)

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

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);

PARAMETERS

names The address of a pointer.
node A handle to a device node.

DESCRIPTION

These functions extract information associated with a device node.

RETURN VALUES

The di_binding_name() 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 di_bus_addr() 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 di_compatible_names() is the number of compatible names.
names is updated to point to a buffer contained within the snapshot. The buffer
contains a concatenation of null-terminated strings, for example:

<name1>/0<name2>/0...<namen>/0

See the discussion of generic names in Writing Device Drivers for a description of how
compatible names are used by Solaris to achieve driver binding for the node.

The di_devid() function returns the device ID for node, if it is registered. Otherwise,
a null pointer is returned. Interfaces in the libdevid(3LIB) 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.
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Evolving (di_devid() is obsolete)</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**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*
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 **di_child_node()** function obtains a handle to the first child of **node**. If no child node exists in the snapshot, **DI_NODE_NIL** is returned and **errno** is set to **ENXIO** or **ENOTSUP**.

The **di_parent_node()** function obtains a handle to the parent node of **node**. If no parent node exists in the snapshot, **DI_NODE_NIL** is returned and **errno** is set to **ENXIO** or **ENOTSUP**.

The **di_sibling_node()** function obtains a handle to the next sibling node of **node**. If no next sibling node exists in the snapshot, **DI_NODE_NIL** is returned and **errno** is set to **ENXIO** or **ENOTSUP**.

The **di_drv_first_node()** function obtains a handle to the first node associated with the driver specified by **drv_name**. If there is no such driver, **DI_NODE_NIL** is returned with **errno** set to **EINVAL**. If the driver exists but there is no node associated with this driver, **DI_NODE_NIL** is returned and **errno** is set to **ENXIO** or **ENOTSUP**.

The **di_drv_next_node()** function returns a handle to the next node bound to the same driver. If no more nodes exist, **DI_NODE_NIL** is returned.

Upon successful completion, a handle is returned. Otherwise, **DI_NODE_NIL** is returned and **errno** is set to indicate the error.
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:

<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>

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
cce [ flag... ] file... -ldevinfo [ library... ]
#include <libdevinfo.h>

char *di_devfs_path(di_node_t node);
void di_devfs_path_free(char *path_buf);

PARAMETERS
node Handle to a device node in the snapshot.
path_buf Pointer returned by di_devfs_path().

DESCRIPTION
The di_devfs_path() function generates the physical path of the device node. 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 node.

ERRORS
The di_devfs_path() function will fail if:
EINVAL The node 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>
<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>

SEE ALSO
malloc(3C), libdevinfo(3DEVINFO), attributes(5)

Writing Device Drivers
**NAME**

`di_init, di_fini` – create and destroy a snapshot of kernel device tree

**SYNOPSIS**

```c
#include <libdevinfo.h>
di_node_t di_init(const char *phys_path, uint_t flags);
void di_fini(di_node_t root);
```

**PARAMETERS**

- `flags` - 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 `flags` is 0, the snapshot contains only a single node without properties or minor nodes.

- `phys_path` - Physical path of the root node of the snapshot. See `di_devfs_path(3DEVINFO)`.

- `root` - 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 root node. The caller specifies the contents of the snapshot by providing `flag` and `phys_path`.

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 `phys_path` is not present in the system, or the `devinfo(7D)` driver is not installed properly.
- `EINVAL` - Either `phys_path` is incorrectly formed or the `flags` 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:

```c
/ *
* Code to print all device tree node names
*/
```
EXAMPLE 1 Using the libdevinfo() Interfaces To Print All Device Tree Node Names
(Continued)

```c
#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:

```c
/*@ */
*/ Code to print physical path of scsi disks */
```
EXAMPLE 2 Using the libdevinfo() Interfaces To Print The Physical Path Of SCSI Disks  
(Continued)

```c
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:

<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>

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

Writing Device Drivers
di_minor_devt(3DEVINFO)

NAME   | di_minor_devt, di_minor_name, di_minor_nodetype, di_minor_spectype -- return libdevinfo minor node information

SYNOPSIS | cc [ flag... ] file... -ldevinfo [ library... ]

   include <libdevinfo.h>

   dev_t di_minor_devt(di_minor_t minor);
   char *di_minor_name(di_minor_t minor);
   char *di_minor_nodetype(di_minor_t minor);
   int di_minor_spectype(di_minor_t minor);

PARAMETERS | minor   A handle to minor data node.

DESCRIPTION | These functions return libdevinfo minor node information.

RETURN VALUES | The di_minor_name() function returns the minor name. See
ddi_create_minor_node(9F) for a description of the name parameter.

   The di_minor_devt() function returns the dev_t value of the minor node that is
   specified by SYS V ABI. See getmajor(9F), getminor(9F), and
ddi_create_minor_node(9F) for more information.

   The di_minor_spectype() function returns the spec_type of the file, either
   S_IFCHR or S_IFBLK. See ddi_create_minor_node(9F) for a description of the
   spec_type parameter.

   The di_minor_nodetype() function returns the minor node_type of the minor node.
   See ddi_create_minor_node(9F) for a description of the node_type parameter.

ERRORS | No errors 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>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO | attributes(5), ddi_create_minor_node(9F), getmajor(9F), getminor(9F)

Writing Device Drivers
di_minor_next(3DEVINFO)

NAME  
di_minor_next – libdevinfo minor node traversal functions

SYNOPSIS  
cc [ flag... ] file... -ldevinfo [ library... ]
#include <libdevinfo.h>

    di_minor_t di_minor_next(di_node_t node, di_minor_t minor);

PARAMETERS  
minor  Handle to the current minor node or DI_MINOR_NIL.
node   Device node with which the minor node is associated.

DESCRIPTION  
The di_minor_next() function returns a handle to the next minor node for the
device node node. If minor 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 errno is set to indicate the error.

ERRORS  
The di_minor_next() function will fail if:

EINVAL           Invalid argument.
ENOTSUP          Minor node information is not available in snapshot.
ENXIO            End of minor node list.

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>

SEE ALSO  
libdevinfo(3DEVINFO), attributes(5)

Writing Device Drivers
NAME
di_prom_init, di_prom_fini – create and destroy a handle to the PROM device information

SYNOPSIS
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);

PARAMETERS
ph 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>

SEE ALSO
di_nodeid(3DEVINFO), di_prom_prop_next(3DEVINFO),
di_prom_prop_lookup_bytes(3DEVINFO), libdevinfo(3DEVINFO),
malloc(3C), attributes(5), openprom(7D)
NAME
di_prom_prop_data, di_prom_prop_next, di_prom_prop_name – access PROM device information

SYNOPSIS
c++ [ flag... ] file... -ldevinfo [ library... ]
#include <libdevinfo.h>

di_prom_prop_t di_prom_prop_next(di_prom_handle_t ph, di_node_t node, di_prom_prop_t prom_prop);
char *di_prom_prop_name(di_prom_prop_t prom_prop);
int di_prom_prop_data(di_prom_prop_t prom_prop, uchar_t **prop_data);

PARAMETERS
node Handle to a device node in the snapshot of kernel device tree.
ph PROM handle
prom_prop Handle to a PROM property.
prop_data Address of a pointer.

DESCRIPTION
The di_prom_prop_next() function obtains a handle to the next property on the PROM property list associated with node. If prom_prop is DI_PROM_PROP_NIL, the first property associated with node is returned.

The di_prom_prop_name() function returns the name of the prom_prop property.

The di_prom_prop_data() function returns the value of the prom_prop property. The return value is a non-negative integer specifying the size in number of bytes in prop_data.

All memory allocated by these functions is managed by the library and must not be freed by the caller.

RETURN VALUES
The di_prom_prop_data() function returns the number of bytes in prop_data and prop_data 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 di_prom_prop_name() function returns a pointer to a string that contains the name of prom_prop.

The di_prom_prop_next() function returns a handle to the next PROM property. DI_PROM_PROP_NIL is returned if no additional properties exist.

ERRORS
See openprom(7D) for a description of possible errors.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
di_prom_prop_data(3DEVINFO)

<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>

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**
```c
#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);
```

**PARAMETERS**
- **node** Handle to device node in snapshot created by `di_init(3DEVINFO)`.
- **ph** Handle returned by `di_prom_init(3DEVINFO)`.
- **prop_data** 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.
- **prop_name** 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 `prop_data` 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 `prop_data` 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 `errno` 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 `prop_data`.
  - For `di_prom_prop_lookup_ints()`, the number of entries is the number of integers contained in the buffer pointed to by `prop_data`.
  - 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:

<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>

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

Writing Device Drivers
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
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);

PARAMETERS
prop Handle to a property returned by di_prop_next(3DEVINFO).
prop_data 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.
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:

- **DI_PROP_TYPE_BOOLEAN**: 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.
- **DI_PROP_TYPE_INT**: Use `di_prop_ints()` to access property data.
- **DI_PROP_TYPE_INT64**: Use `di_prop_int64()` to access property data.
- **DI_PROP_TYPE_STRING**: Use `di_prop_strings()` to access property data.
- **DI_PROP_TYPE_BYTE**: Use `di_prop_bytes()` to access property data.
- **DI_PROP_TYPE_UNKNOWN**: Use `di_prop_bytes()` to access property data. Since the type of property is unknown, the caller is responsible for interpreting the contents of the data.
- **DI_PROP_TYPE_UNDEF_IT**: 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_prop_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:

- **EINVAL**: Invalid argument. For example, the property type does not match the interface.

**ATTRIBUTES**

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

- `di_prom_prop_lookup_bytes(3DEVINFO)`
- `di_prop_next(3DEVINFO)`
- `libdevinfo(3DEVINFO)`
- `attributes(5)`

*Writing Device Drivers*
di_prop_lookup_bytes(3DEVINFO)

NAME  di_prop_lookup_bytes, di_prop_lookup_ints, di_prop_lookup_int64,
       di_prop_lookup_strings – search for a property

SYNOPSIS  cc [ flg... ] flk... -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);

PARAMETERS

dev  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.

node  Handle to the device node with which the property is associated.

prop_data  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.

prop_name  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 prop_data 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 errno 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_*().

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>

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

Writing Device Drivers
di_prop_next(3DEVINFO)

NAME  
di_prop_next – libdevinfo property traversal function

SYNOPSIS  
cc [ flag... ] file... -ldevinfo [ library... ]  
#include <libdevinfo.h>

di_prop_t di_prop_next(di_node_t node, di_prop_t prop);

PARAMETERS  
node Handle to a device node.

prop Handle to a property.

DESCRIPTION  
The di_prop_next() function returns a handle to the next property on the property
list. If prop 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>
<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>

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

Writing Device Drivers
### DisconnectToServer(3DMI)

#### NAME
DisconnectToServer - disconnect from a DMI service provider

#### SYNOPSIS
```
cc [ flag ... ] file ... -ldmici -ldmimi [ library ... ]
#include <dmi/api.hh>

bool_t DisconnectToServer(DmiRpcHandle *dmi_rpc_handle);
```

#### DESCRIPTION
The DisconnectToServer() function disconnects a management application or a component instrumentation from a DMI service provider.

#### RETURN VALUES
The ConnectToServer() function returns TRUE if successful, otherwise FALSE.

#### 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>
</tbody>
</table>

#### SEE ALSO
ConnectToServer(3DMI), attributes(5)
NAME
di_walk_minor - traverse libdevinfo minor nodes

SYNOPSIS
ec [ 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);

PARAMETERS
arg Pointer to caller-specific user data.
flag Specify 0. Reserved for future use.
minor The minor node visited.
minor_nodetype 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 minor_node types. See
ddi_create_minor_node(9F).
node The device node with which to the minor node is associated.
root Root of subtree to visit.

DESCRIPTION
The di_walk_minor() function visits all minor nodes attached to device nodes in a
subtree rooted at root. For each minor node that matches minor_nodetype, the
caller-supplied function minor_callback() is invoked. The walk terminates immediately
when minor_callback() 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>
<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>

SEE ALSO
di_minor_nodetype(3DEVINFO), libdevinfo(3DEVINFO), attributes(5),
ddi_create_minor_node(9F)
Writing Device Drivers

di_walk_minor(3DEVINFO)
The `di_walk_node()` function visits all nodes in the subtree rooted at `root`. For each node found, the caller-supplied function `node_callback()` is invoked. The return value of `node_callback()` specifies subsequent walking behavior.

**PARAMETERS**
- `arg` Pointer to caller-specific data.
- `flag` Specifies walking order, either `DI_WALK_CLDFIRST` (depth first) or `DI_WALK_SIBFIRST` (breadth first). `DI_WALK_CLDFIRST` is the default.
- `node` The node being visited.
- `root` The handle to the root node of the subtree to visit.

**RETURN VALUES**
Upon successful completion, `di_walk_node()` returns `0`. Otherwise, `-1` is returned and `errno` is set to indicate the error.

The `node_callback()` function can return one of the following:
- `DI_WALK_CONTINUE` Continue walking.
- `DI_WALK_PRUNESIB` Continue walking, but skip siblings and their child nodes.
- `DI_WALK_PRUNECHILD` Continue walking, but skip subtree rooted at current node.
- `DI_WALK_TERMINATE` Terminate the walk immediately.

**ERRORS**
The `di_walk_node()` function will fail if:
- `EINVAL` Invalid argument.

**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>

**SEE ALSO**
- `di_init(3DEVINFO)`, `libdevinfo(3DEVINFO)`, `attributes(5)`
- *Writing Device Drivers*
### DmiAddComponent(3DMI)

**NAME**

DmiAddComponent, DmiAddGroup, DmiAddLanguage, DmiDeleteComponent, DmiDeleteGroup, DmiDeleteLanguage – Management Interface database administration functions

**SYNOPSIS**

```c
cc [ flag ... ] file ... -ldmimi -ldmi -lnsl -lrwtool [ library ... ]
#include <dmi/server.h>
#include <dmi/miapi.h>

bool_t DmiAddComponent(DmiAddComponentIN argin, DmiAddComponentOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiAddGroup(DmiAddGroupIN argin, DmiAddGroupOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiAddLanguage(DmiAddLanguageIN argin, DmiAddLanguageOUT*result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiDeleteComponent(DmiDeleteComponentIN argin, DmiDeleteComponentOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiDeleteGroup(DmiDeleteGroupIN argin, DmiDeleteGroupOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiDeleteLanguage(DmiDeleteLanguageIN argin, DmiDeleteLanguageOUT *result, DmiRpcHandle *dmi_rpc_handle);
```

**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 `DmiAddComponent()` 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 `argin` parameter is an instance of a `DmiAddComponentIN` structure containing the following members:

- `DmiHandle_t handle; /* an open session handle */`
- `DmiFileDataList_t *fileData; /* MIF data for component */`

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

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

The `DmiAddLanguage()` 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 `argin` parameter is an instance of a `DmiAddLanguageIN` structure containing the following members:
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:

DmiHandle_t handle; /* an open session handle */
DmiId_t compId; /* component to access */
DmiStringList_t *groups; /* group list */
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
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWsasdk</td>
</tr>
<tr>
<td>MT-level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

attributes(5)
NAME
DmiAddRow, DmiDeleteRow, DmiGetAttribute, DmiGetMultiple, DmiSetAttribute,
DmiSetMultiple – Management Interface operation functions

SYNOPSIS
cc [ flag ... ] file ... -ldmimi -ldmi -lnsl -lrwtool [ library ... ]
#include <server.h>
#include <miapi.h>

bool_t DmiAddRow(DmiAddRowIN argin, DmiAddRowOUT *result,
                 DmiRpcHandle *dmi_rpc_handle);
bool_t DmiDeleteRow(DmiDeleteRowIN argin, DmiDeleteRowOUT *result,
                    DmiRpcHandle *dmi_rpc_handle);
bool_t DmiGetAttribute(DmiGetAttributeIN argin, DmiGetAttributeOUT
                      *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiGetMultiple(DmiGetMultipleIN argin, DmiGetMultipleOUT
                     *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiSetAttribute(DmiSetAttributeIN argin, DmiSetAttributeOUT
                      *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiSetMultiple(DmiSetMultipleIN argin, DmiSetMultipleOUT
                     *result, DmiRpcHandle *dmi_rpc_handle);

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 DmiAddRow() function adds a row to an existing table. The rowData 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 argin parameter is an instance of a
DmiAddRowIN structure containing the following members:

DmiHandle_t handle;  /* An open session handle */
DmiRowData_t *rowData; /* Attribute values to set */

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

DmiErrorStatus_t  error_status;

DmiDeleteRow() function removes a row from an existing table. The key list must
specify valid keys for a table row. The argin parameter is an instance of a
DmiDeleteRowIN structure containing the following members:

DmiHandle_t handle;  /* An open session handle */
DmiRowData_t *rowData; /* Row to delete */

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

DmiErrorStatus_t  error_status;
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; /* An open session 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:
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_VALUEUNKNOWN

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

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:
DmiAddRow(3DMI)

<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>

SEE ALSO attributes(5)
dmi_error(3DMI)

**NAME**  
dmi_error – print error in string form

**SYNOPSIS**  
cce [ flag ... ] file ... -ldmi -lnsl -lrwtool [ library ... ]  
#include <dmi/dmi_error.hh>

```c
void dmi_error(DmiErrorStatus_t error_status);
```

**DESCRIPTION**  
For the given error_status, the dmi_error() function prints the corresponding error in string form. The function prints "unknown dmi errors" if error_status is invalid.

**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>

**SEE ALSO**  
libdmi(3LIB), attributes(5)
### SYNOPSIS

```bash
cc [ flag ... ] file ... -ldmimi -ldmi -lnsl -lrwtool [ library ... ]
#include <server.h>
#include <miapi.h>
bool_t DmiGetConfig(DmiGetConfigIN argin, DmiGetConfigOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiGetVersion(DmiGetVersionIN argin, DmiGetVersionOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiRegister(DmiRegisterIN argin, DmiRegisterOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiSetConfig(DmiSetConfigIN argin, DmiSetConfigOUT *result, DmiRpcHandle *dmi_rpc_handle);
bool_t DmiUnregister(DmiUnregisterIN argin, DmiUnregisterOUT *result, DmiRpcHandle *dmi_rpc_handle);
```

### 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 `DmiGetConfig()` function retrieves the per-session configuration information. The configuration information consists of a string describing the current language being used for the session. The `argin` parameter is an instance of a `DmiGetConfigIN` structure containing the following member:

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

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

```
DmiErrorStatus_t error_status;
DmiString_t *language; /* current session language */
```

The `DmiGetVersion()` function retrieves information about the Service Provider. The management application uses the `DmiGetVersion()` 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 `argin` parameter is an instance of a `DmiGetVersionIN` structure containing the following member:

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

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

```
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 */
```
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:

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

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

```c
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:

```c
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:

```c
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:

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

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

```c
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:
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:

<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>

**SEE ALSO**

attributes(5)
DmiListAttributes(3DMI)

NAME
DmiListAttributes, DmiListClassNames, DmiListComponents,
DmiListComponentsByClass, DmiListGroups, DmiListLanguages – Management
Interface listing functions

SYNOPSIS
ce [ flag ... ] file ... -ldmimi -ldmi -lnsl -lruntool [ library ... ]
#include <server.h>
#include <miapi.h>

bool_t DmiListAttributes(DmiListAttributesIN argin,
                        DmiListAttributesOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListClassNames(DmiListClassNamesIN argin,
                         DmiListClassNamesOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListComponents(DmiListComponentsIN argin,
                         DmiListComponentsOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListComponentsByClass(DmiListComponentsByClassIN argin,
                                DmiListComponentsByClassOUT *result, DmiRpcHandle
                                *dmi_rpc_handle);

bool_t DmiListGroup(DmiListGroupIN argin, DmiListGroupOUT
                        *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiListLanguages(DmiListLanguagesIN argin,
                        DmiListLanguagesOUT *result, DmiRpcHandle *dmi_rpc_handle);

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 DmiListComponents() function retrieves the names and (optionally) the
description of components in a system. Use this to interrogate a system to determine
what components are installed. The argin parameter is an instance of a
DmiListComponentsIN 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,
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 */

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

DmiErrorStatus_t error_status;
DmiComponentList_t *reply;    /* list of components */
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:
DmiListAttributes(3DMI)

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 DmiListGroups() 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 DmiListGroupsIN 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 DmiListGroupsOUT 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.
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 *arg* parameter is an instance of a DmiListAttributesIN structure containing the following members:

- DmiHandle_t handle; /* An open session handle */
- DmiRequestMethod_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 access */
- DmiId_t attribId; /* Attribute to start with, refer to requestMode */

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

- DmiErrorStatus_t error_status;
- DmiAttributeList_t *reply; /* List of attributes */

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
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:

<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>

**SEE ALSO**

attributes(5)
DmiRegisterCi, DmiUnRegisterCi, DmiOriginateEvent – Service Provider functions for components

**SYNOPSIS**

```c
#include <server.h>
#include <ciapi.h>

extern bool_t DmiRegisterCi(DmiRegisterCiIN argin, DmiRegisterCiOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiUnregisterCi(DmiUnregisterCiIN argin, DmiUnregisterCiOUT *result, DmiRpcHandle *dmi_rpc_handle);

bool_t DmiOriginateEvent(DmiOriginateEventIN argin, DmiOriginateEventOUT *result, DmiRpcHandle *dmi_rpc_handle);
```

**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, `argin` is the parameter passed to initiate an RPC call, `result` is the result of the RPC call, and `dmi_rpc_handle` is an open session RPC handle.

The `DmiRegisterCi()` function registers a callable interface for components that have resident instrumentation code and/or to get the version of the Service Provider.

The `DmiUnRegisterCi()` function communicates to the Service Provider to remove a direct component instrumentation interface from the Service Provider table of registered interfaces.

The `DmiOriginateEvent()` 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 `DmiRegisterCi()` function returns the following possible values:

- DMIERR_NO_ERROR
- DMIERR_ILLEGAL_HANDLE
- DMIERR_OUT_OF_MEMORY
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:

<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>

**SEE ALSO**

attributes(5)
NAME

ea_error – error interface to extended accounting library

SYNOPSIS

cc [flag...] file... -lexacct [library ...]
#include <exacct.h>

int ea_error(void);

DESCRIPTION

The ea_error() function returns the error value of the last failure recorded by the invocation of one of the functions of the extended accounting library, libexacct.

RETURN VALUES

EXR_CORRUPT_FILE A function failed because the file was not a valid exactt 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 ea_set_item(3EXACCT).
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 errno variable contains the error value set by the underlying call.
EXR_UNKN_VERSION The file referred to by name uses an exactt file version that cannot be processed by this library.

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>

SEE ALSO

read(2), libexacct(3LIB), attributes(5)
**ea_open(3EXACCT)**

**NAME**
ea_open, ea_close – open or close exact files

**SYNOPSIS**
cc [flag...] file... -lexacct [library...]
#include <exacct.h>

```c
int ea_open(ea_file_t *ef, char *name, char *creator, int aflags, int oflags, mode_t mode);
int ea_close(ea_file_t *ef);
```

**DESCRIPTION**
The `ea_open()` function provides structured access to exact files. The `aflags` argument contains the appropriate `exact` flags necessary to describe the file. The `oflags` and `mode` arguments contain the appropriate flags and mode to open the file; see `<fcntl.h>`. If `ea_open()` is invoked with `EO_HEAD` specified in `aflags`, the resulting file is opened with the object cursor located at the first object of the file. If `ea_open()` is invoked with `EO_TAIL` specified in `aflags`, the resulting file is opened with the object cursor positioned beyond the last object in the file. If `EO_NO_VALID_HDR` is set in `aflags` along with `EO_HEAD`, the initial header record will be returned as the first item read from the file. When creating a file, the `creator` 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 `ea_close()` function closes an open exact file.

**RETURN VALUES**
Upon successful completion, `ea_open()` and `ea_close()` return 0. Otherwise they return −1 and call `ea_error(3EXACCT)` to return the extended accounting error value describing the error.

**ERRORS**
The `ea_open()` and `ea_close()` functions may fail if:

- **EXR_SYSCALL_FAIL**
  A system call invoked by the function failed. The `errno` variable contains the error value set by the underlying call.

The `ea_open()` function may fail if:

- **EXR_CORRUPT_FILE**
The file referred to by `name` is not a valid `exact` file.

- **EXR_NO_CREATOR**
  In the case of file creation, the `creator` argument was NULL. In the case of opening an existing file, a `creator` argument was not NULL and does not match the `creator` item of the `exact` file.

- **EXR_UNKN_VERSION**
The file referred to by `name` uses an `exact` file version that cannot be processed by this library.

**USAGE**
The `exact` 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.
EXAMPLE 1 Open and close exacct file.

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

```c
#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:

<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>

SEE ALSO

ea_error(3EXACCT), ea_pack_object(3EXACCT), ea_set_item(3EXACCT), libexacct(3LIB), attributes(5)
The `ea_pack_object()` function converts `exacct` objects from their in-memory representation to their file representation. It is passed an object pointer that points to the top of an `exacct` object hierarchy representing one or more `exacct` 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 `buf` and `bufsize` parameters can be set to `NULL` and 0 respectively, and the required buffer size will be returned. The resulting packed record can be passed to `putacct(2)` or to `ea_set_item(3EXACCT)` when constructing an object of type `EXT_EXACCT_OBJECT`. 

The `ea_unpack_object()` function reverses the packing process performed by `ea_pack_object()`. A packed buffer passed to `ea_unpack_object()` is unpacked into the original hierarchy of objects. If the unpack operation fails (for example, due to a corrupted or incomplete buffer), it returns `EO_ERROR`; otherwise, the object type of the first object in the hierarchy is returned. If `ea_unpack_object()` is invoked with `flag` equal to `EUP_ALLOC`, it allocates memory for the variable-length data in the included objects. Otherwise, with `flag` equal to `EUP_NOALLOC`, it sets the variable length data pointers within the unpacked object structures to point within the buffer indicated by `buf`. In both cases, `ea_unpack_object()` allocates all the necessary `exacct` objects to represent the unpacked record. The resulting object hierarchy can be freed using `ea_free_object(3EXACCT)` with the same `flag` value.
The `ea_get_creator()` function returns a pointer to a string representing the recorded creator of the exactct file. The `ea_get_hostname()` function returns a pointer to a string representing the recorded hostname on which the exactct file was created. These functions will return `NULL` if their respective field was not recorded in the exactct 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 exactct 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 exactct 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()` function returns the root object of the group. The `ea_pack_object()` function is used to pack the object into a buffer.
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 exact 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 exact object in the file. It returns EO_ERROR if the exact 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 exact object unpacked from the buffer. It returns EO_ERROR if the exact 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:

EXR_SYSCALL_FAIL A system call invoked by the function failed. The errno variable contains the error value set by the underlying call.

EXR_CORRUPT_FILE The file referred to by name is not a valid exact file, or is unparsable, and therefore appears corrupted. This error is also used by ea_unpack_buffer() to indicate a corrupted buffer.

EXR_NO_MEMORY A memory allocation required to complete the operation failed.

EXR_EOF The end of the file has been reached. In the case of ea_previous_record(), the previous record could
not be reached, either because the head of the file was encountered or because the previous record could not be skipped over.

**USAGE**
The exact 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.

**EXAMPLE 1** Open and close exact file.

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

```c
#include <stdio.h>
#include <exact.h>

e_file_t ef;
e_object_t *obj;
...

e_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 */
}

e_close(&ef);
```

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

```c
#include <sys/types.h>
#include <unistd.h>
#include <exact.h>
...

e_file_t ef;
e_object_t obj;

pid_t my_pid;

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

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

Extended Library Functions 143
ea_pack_object(3EXACCT)

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

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

**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>

**SEE ALSO**

read(2), ea_error(3EXACCT), ea_open(3EXACCT), ea_set_item(3EXACCT), libexacct(3LIB), attributes(5)
The ea_alloc() function allocates a block of memory of the requested size. This block can be safely passed to libexacct functions, and can be safely freed by any of the ea_free() functions.

The ea_strdup() function can be used to duplicate a string that is to be stored inside an ea_object_t structure.

The ea_set_item() function assigns the given exact object to be a data item with value set according to the remaining arguments. For buffer-based data values (EXT_STRING, EXT_EXACCT_OBJECT, and EXT_RAW), a copy of the passed buffer is taken. In the case of EXT_EXACCT_OBJECT, the passed buffer should be a packed exact object as returned by ea_pack_object(3EXACCT). Any item assigned with ea_set_item() should be freed with ea_free_item() specifying a flag value of EUP_ALLOC when the item is no longer needed.

The ea_match_object_catalog() function returns TRUE if the exact object specified by obj has a catalog tag that matches the mask specified by catmask.

The ea_attach_to_object() function attaches an object to the given object. The ea_attach_to_group() 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 `ea_free()` function frees a block of memory previously allocated by `ea_alloc()`.

The `ea_strfree()` function frees a string previously copied by `ea_strdup()`.

The `ea_free()` function frees the value fields in the `ea_object_t` indicated by `obj`, if `EUP_ALLOC` is specified. The object itself is not freed. The `ea_free_object()` function frees the specified object and any attached hierarchy of objects. If the `flag` argument is set to `EUP_ALLOC`, `ea_free_object()` will also free any variable-length data in the object hierarchy; if set to `EUP_NOALLOC`, `ea_free_object()` will not free variable-length data. In particular, these flags should correspond to those specified in calls to `ea_unpack_object(3EXACCT)`.

**RETURN VALUES**

The `ea_match_object_catalog()` 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 `ea_error(3EXACCT)`.

**ERRORS**

The `ea_set_item()`, `ea_set_group()`, and `ea_match_object_catalog()` functions may fail if:

- **EXR_SYSCALL_FAIL** A system call invoked by the function failed. The `errno` variable contains the error value set by the underlying call.

- **EXR_INVALID_OBJECT** The passed object is of an incorrect type, for example passing a group object to `ea_set_item()`.

**USAGE**

The exact 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 exact file.

Construct an exact file consisting of a single object containing the current process ID.

```c
#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));
```
EXAMPLE 1 Open and close exacct file.  (Continued)

...  

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>

SEE ALSO  

read(2), ea_error(3EXACCT), ea_open(3EXACCT), ea_pack_object(3EXACCT), libexacct(3LIB), attributes(5)
elf32_checksum(3ELF)

NAME    elf32_checksum, elf64_checksum – return checksum of elf image

SYNOPSIS cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

long elf32_checksum(Elf *elf);
long elf64_checksum(Elf *elf);

DESCRIPTION The elf32_checksum() function returns a simple checksum of selected sections of
the image identified by elf. The value is typically used as the .dynamic tag
DT_CHECKSUM, 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 strip(1).

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

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>

SEE ALSO elf(3ELF), elf_version(3ELF), gelf(3ELF), libelf(3LIB), attributes(5)
**NAME**
elf32_fsize, elf64_fsize – return the size of an object file type

**SYNOPSIS**
```c
#include <libelf.h>
size_t elf32_fsize(Elf_Type type, size_t count, unsigned ver);
size_t elf64_fsize(Elf_Type type, size_t count, unsigned ver);
```

**DESCRIPTION**
elf32_fsize() gives the size in bytes of the 32-bit file representation of count data objects with the given type. The library uses version ver to calculate the size. See elf(3ELF) and elf_version(3ELF).

Constant values are available for the sizes of fundamental types:

<table>
<thead>
<tr>
<th>Elf_Type</th>
<th>File Size</th>
<th>Memory Size</th>
</tr>
</thead>
<tbody>
<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>ELF_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>
</tbody>
</table>

elf32_fsize() returns 0 if the value of type or ver is unknown. See elf32_xlatetof(3ELF) for a list of the type values.

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

**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>

**SEE ALSO**
ellf(3ELF), elf32_xlatetof(3ELF), elf_version(3ELF), libelf(3LIB), attributes(5)
elf32_getehdr(3ELF)

NAME
elf32_getehdr, elf32_newehdr, elf64_getehdr, elf64_newehdr – retrieve class-dependent object file header

SYNOPSIS
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);

DESCRIPTION
For a 32-bit class file, elf32_getehdr() returns a pointer to an ELF header, if one is available for the ELF descriptor elf. If no header exists for the descriptor, elf32_newehdr() allocates a clean one, but it otherwise behaves the same as elf32_getehdr(). It does not allocate a new header if one exists already. If no header exists for elf32_getehdr(), one cannot be created for elf32_newehdr(), a system error occurs, the file is not a 32-bit class file, or elf 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:

unsigned char e_ident[8];
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_Half e_ehsize;
Elf32_Half e_phentsize;
Elf32_Half e_phnum;
Elf32_Half e_shentsize;
Elf32_Half e_shnum;
Elf32_Half e_shstrndx;

elf32_newehdr() automatically sets the ELF_F_DIRTY bit. See elf_flagdata(3ELF). A program may use elf_getident() to inspect the identification bytes from a file.

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>
elf32_getehdr(3ELF)

SEE ALSO elf(3ELF), elf_begin(3ELF), elf_flagdata(3ELF), elf_getident(3ELF), libelf(3LIB), attributes(5)
### elf32_getphdr(3ELF)

**NAME**
elf32_getphdr, elf32_newphdr, elf64_getphdr, elf64_newphdr – retrieve class-dependent program header table

**SYNOPSIS**
```
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);
```

**DESCRIPTION**
For a 32-bit class file, `elf32_getphdr()` returns a pointer to the program execution header table, if one is available for the ELF descriptor `elf`.

`elf32_newphdr()` allocates a new table with `count` entries, regardless of whether one existed previously, and sets the ELF_P_DIRTY bit for the table. See `elf_flagdata(3ELF)`. Specifying a zero `count` deletes an existing table. Note this behavior differs from that of `elf32_newehdr()` allowing a program to replace or delete the program header table, changing its size if necessary. See `elf32_getehdr(3ELF)`. If no program header table exists, the file is not a 32-bit class file, an error occurs, or `elf` is NULL, both functions return a null pointer. Additionally, `elf32_newphdr()` returns a null pointer if `count` is 0.

The table is an array of Elf32_Phdr structures, each of which includes the following members:

```
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;
```

The Elf64_Phdr structures include the following members:

```
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;
```

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

<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>

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

NAME  elf32_getshdr, elf64_getshdr – retrieve class-dependent section header

SYNOPSIS  cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

Elf32_Shdr *elf32_getshdr(Elf_Scn *scn);
Elf64_Shdr *elf64_getshdr(Elf_Scn *scn);

DESCRIPTION  For a 32-bit class file, elf32_getshdr() returns a pointer to a section header for the
section descriptor scn. Otherwise, the file is not a 32-bit class file, scn was NULL, or an
error occurred; elf32_getshdr() then returns NULL.

The elf32_getshdr header includes the following members:

- 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;

while the elf64_getshdr header includes the following members:

- 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;

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 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>
elf32_gethdr(3ELF)

SEE ALSO
elf(3ELF), elf_flagdata(3ELF), elf_getscn(3ELF), elf_strptr(3ELF),
libelf(3LIB), attributes(5)
elf32_xlatetof(3ELF)

NAME  elf32_xlatetof, elf32_xlatetom, elf64_xlatetof, elf64_xlatetom – class-dependent data translation

SYNOPSIS  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);

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. src is a pointer to the source buffer that holds the original data; dst is a pointer to a destination buffer that will hold the translated copy. encode 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 e_ident[EI_DATA] entry (see elf_getident(3ELF)). If the data can be translated, the functions return dst. 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:

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.

Translation routines allow the source and destination buffers to coincide. That is, dst→d_buf may equal src→d_buf. Other cases where the source and destination buffers overlap give undefined behavior.
elf32_xlatetof(3ELF)

<table>
<thead>
<tr>
<th>Elf_Type</th>
<th>32-Bit Memory Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF_T_ADDR</td>
<td>Elf32_Addr</td>
</tr>
<tr>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>ELF_T_DYN</td>
<td>Elf32_Dyn</td>
</tr>
<tr>
<td>ELF_T_ENHDR</td>
<td>Elf32_Ehdr</td>
</tr>
<tr>
<td>ELF_T_HALF</td>
<td>Elf32_Half</td>
</tr>
<tr>
<td>ELF_T_OFF</td>
<td>Elf32_Off</td>
</tr>
<tr>
<td>ELF_T_PHDR</td>
<td>Elf32_Phdr</td>
</tr>
<tr>
<td>ELF_T_REL</td>
<td>Elf32_Rel</td>
</tr>
<tr>
<td>ELF_T_RELX</td>
<td>Elf32_Relx</td>
</tr>
<tr>
<td>ELF_T_SHDR</td>
<td>Elf32_Shdr</td>
</tr>
<tr>
<td>ELF_T_SWORD</td>
<td>Elf32_Sword</td>
</tr>
<tr>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>ELF_T_WORD</td>
<td>Elf32_Word</td>
</tr>
</tbody>
</table>

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:

<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>

**SEE ALSO**

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

NAME
elf – object file access library

SYNOPSIS
cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

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 ELF descriptor. That is, when the program starts working with a file, elf_begin(3ELF) 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 section descriptors to manipulate the sections of the file (see elf_getscn(3ELF)). 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, data descriptors 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 class, 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>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
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.

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 `elf_version(3ELF)` to pass its version to the library, thus establishing the working version 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, `elf_begin()` 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 `elf_cnt1(3ELF)` to retrieve all necessary data from the file, allowing the program to close the file descriptor and reuse it.

Although the `elf_begin()` interfaces are convenient and efficient for many programs, they might be inappropriate for some. In those cases, an application may invoke the `elf32_xlatetom(3ELF)` or `elf32_xlatetof(3ELF)` 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.

- **elf**
  - **name**
    - These class-independent names perform some service, `name`, for the program.
  - **elf32_name**
    - Service names with an embedded class, 32 here, indicate they work only for the designated class of files.
  - **Elf_Type**
    - Data types can be class-independent as well, distinguished by `Type`.
  - **Elf32_Type**
    - Class-dependent data types have an embedded class name, 32 here.
  - **ELF_C_CMD**
    - Several functions take commands that control their actions. These values are members of the `Elf_Cmd` enumeration; they range from zero through `ELF_C_NUM-1`.
  - **ELF_F_FLAGS**
    - Several functions take flags that control library status and/or actions. Flags are bits that may be combined.
  - **ELF32_FSZ_TYPE**
    - These constants give the file sizes in bytes of the basic ELF types for the 32-bit class of files. See `elf32_fsize()` for more information.
  - **ELF_K_KIND**
    - The function `elf_kind()` identifies the `KIND` of file associated with an ELF descriptor. These values are
members of the Elf_Kind enumeration; they range from zero through ELF_K_NUM−1.

ELF_T_TYPE

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

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.

```c
#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
", cnt,
            (char *)data->d_buf + shdr->sh_name);
    } /* end main */

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

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>

SEE ALSO

ar(3HEAD), elf32_checksum(3ELF), elf32_fsize(3ELF),
elf32_getshdr(3ELF), elf32_xlatetoI(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), ifcompile(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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>M32</td>
<td>AT&amp;T WE 32100</td>
</tr>
<tr>
<td>SPARC</td>
<td>SPARC</td>
</tr>
</tbody>
</table>
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.

```c
#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:

```c
#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.
elf_begin(3ELF)

NAME  elf_begin, elf_end, elf_memory, elf_next, elf_rand – process ELF object files

SYNOPSIS

cc [ flag... ] file ... -l elf [ library ... ]
#include <libelf.h>

Elf *elf_begin(int fd, Elf_Cmd cmd, Elf *ref);

int elf_end(Elf *elf);

Elf *elf_memory(char *image, size_t sz);

Elf_Cmd elf_next(Elf *elf);

size_t elf_rand(Elf *elf, size_t offset);

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. fd is an open file descriptor that elf_begin() uses for reading or writing. elf 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.

cmd may have the following values:

ELF_C_NULL  When a program sets cmd to this value, elf_begin() returns a null pointer, without opening a new descriptor. ref 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 cmd to this value. Depending on the value of ref, this command examines archive members or entire files. Three cases can occur.

First, if ref 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 ref 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 ref appropriately (except for the initial member, which elf_begin() prepares; see the example below). In this case, fd is the same file descriptor used for the parent archive.
Finally, if ref is a non-null ELF descriptor that is not an archive, elf_begin() increments the number of activations for the descriptor and returns ref, without allocating a new descriptor and without changing the descriptor’s read/write permissions. To terminate the descriptor for ref, the program must call elf_end() once for each activation. See the examples below for more information.

**ELF_C_RDWR**

This command duplicates the actions of ELF_C_READ and additionally allows the program to update the file image (see elf_update(3ELF)). That is, using ELF_C_READ gives a read-only view of the file, while ELF_C_RDWR lets the program read and write the file. ELF_C_RDWR is not valid for archive members. If ref is non-null, it must have been created with the ELF_C_RDWR command.

**ELF_C_WRITE**

If the program wishes to ignore previous file contents, presumably to create a new file, it should set cmd to this value. ref 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_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_next(). 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.

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_ctl() to let it reuse file descriptors. After calling elf_ctl() 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.

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

```c
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.

```c
elel_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).

```c
#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.

```c
elel_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);
ifdels = open("path/name", O_RDWR);
elf = elf_begin(ifdels, 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:

<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>

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
cc [ flag ... ] file ... -elf [ library ... ]
#include <libelf.h>

int elf_cntl(Elf *elf, Elf_Cmd cmd);

DESCRIPTION
elf_cntl() instructs the library to modify its behavior with respect to an ELF descriptor, elf. 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 elf. 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 cmd argument tells what actions to take and may have the following values:

ELF_C_FDDONE This value tells the library not to use the file descriptor associated with elf. 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.

If elf_cntl() succeeds, it returns 0. Otherwise elf was NULL or an error occurred, and the function returns −1.

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>

SEE ALSO
elf(3ELF), elf_begin(3ELF), elf_getdata(3ELF), elf_rawfile(3ELF), libelf(3LIB), attributes(5)
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
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

const char *elf_errmsg(int err);

int elf_errno(void);

DESCRIPTION
If an ELF library function fails, a program can call elf_errno() 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 elf_errmsg() function takes an error number, err, and returns a null-terminated error message (with no trailing new-line) that describes the problem. A zero err 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 err 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, elf_errmsg() 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 elf_errmsg() function.

The following fragment clears the internal error number and checks it later for errors. Unless an error occurs after the first call to elf_errno(), the next call will return 0.

(void)elf_errno( );
/* processing ... */
while (more_to_do)
{
  if ((err = elf_errno( )) != 0)
  {
    /* print msg */
    msg = elf_errmsg(err);
  }
}

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>

SEE ALSO
elf(3ELF), libelf(3LIB), attributes(5)
elf_fill(3ELF)

NAME  elf_fill – set fill byte

SYNOPSIS  cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

void elf_fill(int fill);

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 fill 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>
<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>

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 not fill holes.
These functions manipulate the flags associated with various structures of an ELF file. Given an ELF descriptor (*elf*), a data descriptor (*data*), or a section descriptor (*scn*), 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.

`cmd` may have the following values:

**ELF_C_CLR** The functions clear the bits that are asserted in `flags`. Only the non-zero bits in `flags` are cleared; zero bits do not change the status of the descriptor.

**ELF_C_SET** The functions set the bits that are asserted in `flags`. Only the non-zero bits in `flags` are set; zero bits do not change the status of the descriptor.

Descriptions of the defined `flags` bits appear below:

**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 `elf_flagphdr()` with this bit set in `flags` and `cmd` equal to `ELF_C_SET`. A later call to `elf_update()` 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 `elf_flagelf()` and applies to the entire file associated with the descriptor.

When a flag bit is set for an item, it affects all the subitems as well. Thus, for example, if the program sets the `ELF_F_DIRTY` bit with `elf_flagelf()`, the entire logical file is “dirty.”
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:

```c
/* 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:

<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>

SEE ALSO

`elf(3ELF), elf32_getehdr(3ELF), elf_getdata(3ELF), elf_update(3ELF), attributes(5)`
elf_getarhdr - retrieve archive member header

SYNOPSIS

cc [ flag ... ] file ... -lelf [ library...] 
#include <libelf.h>

Elf_Arhdr *elf_getarhdr(Elf *elf);

DESCRIPTION

elf_getarhdr() returns a pointer to an archive member header, if one is available for the ELF descriptor elf. Otherwise, no archive member header exists, an error occurred, or elf was null; elf_getarhdr() then returns a null value. The header includes the following members.

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;

An archive member name, available through ar_name, is a null-terminated string, with the ar format control characters removed. The ar_rawname 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 (ar_name) 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 ar_name to the appropriate value.

Under some error conditions, a member’s name might not be available. Although this causes the library to set ar_name to a null pointer, the ar_rawname member will be set as usual.

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>
</tbody>
</table>
elf_getarhdr(3ELF)

<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>

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
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>
Elf_Arsym *elf_getarsym(Elf *elf, size_t *ptr);

DESCRIPTION
The elf_getarsym() function returns a pointer to the archive symbol table, if one is available for the ELF descriptor elf. Otherwise, the archive doesn’t have a symbol table, an error occurred, or elf was null; elf_getarsym() then returns a null value. The symbol table is an array of structures that include the following members.

char *as_name;
size_t as_off;
unsigned long as_hash;

These members have the following semantics:

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 as_off may be passed as arguments to elf_rand(). See elf_begin(3ELF) to access the desired archive member.

as_hash
This is a hash value for the name, as computed by elf_hash().

If ptr is non-null, the library stores the number of table entries in the location to which ptr 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 as_name, a zero value for as_off, and ~0UL for as_hash.

The hash value returned is guaranteed not to be the bit pattern of all ones (~0UL).

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>

SEE ALSO
ar(3HEAD), elf(3ELF), elf_begin(3ELF), elf_getarhdr(3ELF),
elf_hash(3ELF), libelf(3LIB), attributes(5)
elf_getbase(3ELF)

NAME    elf_getbase – get the base offset for an object file

SYNOPSIS cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

off_t elf_getbase(EI cf *elf);

DESCRIPTION The elf_getbase() function returns the file offset of the
first byte of the file or
archive member associated with elf, if it is known or obtainable, and −1 otherwise. A
null elf is allowed, to simplify error handling; the return value in this case is −1. The
base offset of an archive member is the beginning of the member’s information, not the
beginning of the archive member header.

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>

SEE ALSO ar(3HEAD), elf(3ELF), elf_begin(3ELF), libelf(3LIB), attributes(5)
elf_getdata, elf_newdata, elf_rawdata – get section data

#include <libelf.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);

DESCRIPTION

These functions access and manipulate the data associated with a section descriptor, *scn*. 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 *elf_getdata()* function lets a program step through a section’s data list. If the incoming data descriptor, *data*, is null, the function returns the first buffer associated with the section. Otherwise, *data* should be a data descriptor associated with *scn*, and the function gives the program access to the next data element for the section. If *scn* is null or an error occurs, *elf_getdata()* returns a null pointer.

The *elf_getdata()* function translates the data from file representations into memory representations (see *elf32_xlatetof*(3ELF)) and presents objects with memory data types to the program, based on the file’s *class* (see *elf*(3ELF)). The working library version (see *elf_version*(3ELF)) specifies what version of the memory structures the program wishes *elf_getdata()* to present.

The *elf_newdata()* 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 (*d_type* below) is set to *ELF_T_BYTE*, and the version (*d_version* 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 *ELF_F_DIRTY* bit for the section’s data (see *elf_flagdata*(3ELF)). If *scn* is null or an error occurs, *elf_newdata()* returns a null pointer.

The *elf_rawdata()* function differs from *elf_getdata()* 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 *elf_cntl*(3ELF)) the file descriptor associated with *elf* before the initial raw operation, because *elf_rawdata()* might read the data from the file to ensure it doesn’t interfere with *elf_getdata()*.

See *elf_rawfile*(3ELF) for a related facility that applies to the entire file. When *elf_getdata()* provides the right translation, its use is recommended over *elf_rawdata()*.

If *scn* is null or an error occurs, *elf_rawdata()* returns a null pointer.

The Elf_Data 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.

d_buf
  A pointer to the data buffer resides here. A data element with no data has a null pointer.

d_type
  This member’s value specifies the type of the data to which d_buf points. A section’s type determines how to interpret the section contents, as summarized below.

d_size
  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 SHT_NOBITS below for more information.)

d_off
  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.

d_align
  This member holds the buffer’s required alignment, from the beginning of the section. That is, d_off 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.

d_version
  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.
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)`.

<table>
<thead>
<tr>
<th>Section Type</th>
<th>Elf_Type</th>
<th>32-bit Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHT_DYNAMIC</td>
<td>ELF_T_DYN</td>
<td>Elf32_Dyn</td>
</tr>
<tr>
<td>SHT_DYNSYM</td>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>SHT_FINI_ARRAY</td>
<td>ELF_T_ADDR</td>
<td>Elf32_Addr</td>
</tr>
<tr>
<td>SHT_GROUP</td>
<td>ELF_T_WORD</td>
<td>Elf32_Word</td>
</tr>
<tr>
<td>SHT_HASH</td>
<td>ELF_T_WORD</td>
<td>Elf32_Word</td>
</tr>
<tr>
<td>SHT_INIT_ARRAY</td>
<td>ELF_T_ADDR</td>
<td>Elf32_Addr</td>
</tr>
<tr>
<td>SHT_NOBITS</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_NOTE</td>
<td>ELF_T_NOTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_NULL</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>SHT_PREINIT_ARRAY</td>
<td>ELF_T_ADDR</td>
<td>Elf32_Addr</td>
</tr>
<tr>
<td>SHT_PROGBITS</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_REL</td>
<td>ELF_T_REL</td>
<td>Elf32_Rel</td>
</tr>
<tr>
<td>SHT_RELA</td>
<td>ELF_T_RELA</td>
<td>Elf32_Reela</td>
</tr>
<tr>
<td>SHT_SYMTAB</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_SYMTAB</td>
<td>ELF_T_SYM</td>
<td>Elf32_Sym</td>
</tr>
<tr>
<td>SHT_SUNW_comdat</td>
<td>ELF_T_BYTE</td>
<td>unsigned char</td>
</tr>
<tr>
<td>SHT_SUNW_move</td>
<td>ELF_T_MOVE</td>
<td>Elf32_Move (sparc)</td>
</tr>
<tr>
<td>SHT_SUNW_move</td>
<td>ELF_T_MOVEP</td>
<td>Elf32_Move (ia32)</td>
</tr>
<tr>
<td>SHT_SUNW_syminfo</td>
<td>ELF_T_SYMINFO</td>
<td>Elf32_Syminfo</td>
</tr>
<tr>
<td>SHT_SUNW_verdef</td>
<td>ELF_T_VDEF</td>
<td>Elf32_Verdef</td>
</tr>
<tr>
<td>SHT_SUNW_verneed</td>
<td>ELF_T_VNEED</td>
<td>Elf32_Verneed</td>
</tr>
</tbody>
</table>

Extended Library Functions 181
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`.

**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.

```c
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.

**ATTRIBUTES**

See `attributes(5)` for descriptions of the following attributes:
**elf_getdata(3ELF)**

<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>

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)
elf_getident(3ELF)

NAME
elf_getident – retrieve file identification data

SYNOPSIS
cc [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

char *elf_getident (Elf *elf, size_t *ptr);

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>
<thead>
<tr>
<th>e_ident Index</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI_MAG0</td>
<td>ELFMAG0</td>
<td>File identification</td>
</tr>
<tr>
<td>EI_MAG1</td>
<td>ELFMAG1</td>
<td></td>
</tr>
<tr>
<td>EI_MAG2</td>
<td>ELFMAG2</td>
<td></td>
</tr>
<tr>
<td>EI_MAG3</td>
<td>ELFMAG3</td>
<td></td>
</tr>
<tr>
<td>EI_CLASS</td>
<td>ELFCCLASSNONE</td>
<td>File class</td>
</tr>
<tr>
<td></td>
<td>ELFCCLASS32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELFCCLASS64</td>
<td></td>
</tr>
<tr>
<td>EI_DATA</td>
<td>ELF DATANONE</td>
<td>Data encoding</td>
</tr>
<tr>
<td></td>
<td>ELF DATA2LSB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELF DATA2MSB</td>
<td></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>
</tbody>
</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
the size of the unmodified area depends on the file type. If \textit{ptr} is non-null, the library stores the number of identification bytes in the location to which \textit{ptr} points. If no data are present, \textit{elf} is null, or an error occurs, the return value is a null pointer, with 0 stored through \textit{ptr}, if \textit{ptr} is non-null.

\textbf{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>

\textbf{SEE ALSO} \texttt{elf(3ELF)}, \texttt{elf32_getehdr(3ELF)}, \texttt{elf_begin(3ELF)}, \texttt{elf_kind(3ELF)}, \texttt{elf_rawfile(3ELF)}, \texttt{libelf(3LIB)}, attributes(5)
elf_getscn, elf_ndxscn, elf_newscn, elf_nextscn – get section information

NAME
dlf_getscn, elf_ndxscn, elf_newscn, elf_nextscn – get section information

SYNOPSIS

cc [ flag ... ] file ... -l elf [ library ... ]
#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);

DESCRIPTION

These functions provide indexed and sequential access to the sections associated with
the ELF descriptor elf. If the program is building a new file, it is responsible for
creating the file’s ELF header before creating sections; see elf32_getehdr(3 ELF).

The elf_getscn() function returns a section descriptor, given an index 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 index 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 elf is NULL,
elf_getscn() returns a null pointer.

The elf_newscn() function creates a new section and appends it to the list for elf.
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 elf 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(3 ELF)). 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(3 ELF)). If the program is
building a new file, it is responsible for creating the file’s ELF header (see
elf32_getehdr(3 ELF)) before creating new sections.

The elf_nextscn() function takes an existing section descriptor, scn, and returns a
section descriptor for the next higher section. One may use a null scn 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, scn, and returns its
section table index. If scn 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.
EXAMPLE 1 A sample of calling elf_getscn() function.  (Continued)

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

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>

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

NAME
elf_hash – compute hash value

SYNOPSIS
cc [ flag ... ] file ... -Welf [ library ... ]
#include <libelf.h>

unsigned long elf_hash(const char *name);

DESCRIPTION
The elf_hash() function computes a hash value, given a null terminated string, name. The returned hash value, h, can be used as a bucket index, typically after computing h mod x to ensure appropriate bounds.

Hash tables may be built on one machine and used on another because elf_hash() uses unsigned arithmetic to avoid possible differences in various machines’ signed arithmetic. Although name is shown as char* above, elf_hash() treats it as unsigned char* to avoid sign extension differences. Using char* eliminates type conflicts with expressions such as elf_hash(name).

ELF files’ symbol hash tables are computed using this function (see elf_getdata(3ELF) and elf32_xlatetof(3ELF)). The hash value returned is guaranteed not to be the bit pattern of all ones (~0UL).

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>

SEE ALSO
elf(3ELF), elf32_xlatetof(3ELF), elf_getdata(3ELF), libelf(3L), attributes(5)
elf_kind(3ELF)

NAME
elf_kind – determine file type

SYNOPSIS
c c [ flag ... ] file ... -l elf [ library ... ]
#include <libelf.h>

Elf_Kind elf_kind(Elf *elf);

DESCRIPTION
This function returns a value identifying the kind of file associated with an ELF descriptor (elf). Defined values are below:

ELF_K_AR
The file is an archive [see ar(3HEAD)]. An ELF descriptor may also be associated with an archive member, not the archive itself, and then elf_kind() identifies the member's type.

ELF_K_COFF
The file is a COFF object file. elf_begin(3ELF) describes the library's handling for COFF files.

ELF_K_ELF
The file is an ELF file. The program may use elf_getident() to determine the class. Other functions, such as elf32_getehdr(), are available to retrieve other file information.

ELF_K_NONE
This indicates a kind of file unknown to the library.

Other values are reserved, to be assigned as needed to new kinds of files. elf should be a value previously returned by elf_begin(). A null pointer is allowed, to simplify error handling, and causes elf_kind() to return ELF_K_NONE.

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>

SEE ALSO
ar(3HEAD), elf(3ELF), elf32_getehdr(3ELF), elf_begin(3ELF), elf_getident(3ELF), libelf(3LIB), attributes(5)
 elf_rawfile(3ELF)

NAME  
elf_rawfile – retrieve uninterpreted file contents

SYNOPSIS  
cc [ flag... ] file ... -l elf [ library ... ]
#include <libelf.h>

char *elf_rawfile(Elf *elf, size_t *ptr);

DESCRIPTION  
The elf_rawfile() 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 elf_rawfile() to retrieve the bytes for an archive member.

A program may not close or disable (see elf_cntl(3ELF)) the file descriptor associated with elf before the initial call to elf_rawfile(), because elf_rawfile() 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.

elf_rawdata() is a related function, providing access to sections within a file. See elf_getdata(3ELF).

If ptr is non-null, the library also stores the file’s size, in 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:

<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>

SEE ALSO  
elf(3ELF), elf32_getehdr(3ELF), elf_begin(3ELF), elf_cntl(3ELF), elf_getdata(3ELF), elf_getident(3ELF), elf_kind(3ELF), libelf(3LIB), attributes(5)

NOTES  
A program that uses elf_rawfile() 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 elf32_getehdr(), elf_getdata(), 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 elf_rawfile() 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.
Multiple calls to elf_rawfile() with the same ELF descriptor return the same value; the library does not create duplicate copies of the file.
elf_strptr(3ELF)

NAME  elf_strptr – make a string pointer
SYNOPSIS  

cc [ flag ... ] file ... -l elf [ library ... ]  
#include <libelf.h>

char *elf_strptr(Elf *elf, size_t section, size_t offset);

DESCRIPTION  
The elf_strptr() function converts a string section offset to a string pointer. elf identifies the file in which the string section resides, and section identifies the section table index for the strings. elf_strptr() normally returns a pointer to a string, but it returns a null pointer when elf is null, section is invalid or is not a section of type SHT_STRTAB, the section data cannot be obtained, offset is invalid, or an error occurs.

EXAMPLES  

EXAMPLE 1 A sample program of calling elf_strptr() function.

A prototype for retrieving section names appears below. The file header specifies the section name string table in the e_shstrndx member. The following code loops through the sections, printing their names.

/* 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
", name? name: "(null)");
}

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>

SEE ALSO  
elf(3ELF), elf32_getshdr(3ELF), elf32_xlatetof(3ELF), elf_getdata(3ELF), libelf(3LIB), attributes(5)

NOTES  
A program may call elf_getdata() to retrieve an entire string table section. For some applications, that would be both more efficient and more convenient than using elf_strptr().
elf_update – update an ELF descriptor

SYNOPSIS
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

off_t elf_update(Elf *elf, Elf_Cmd cmd);

DESCRIPTION
The elf_update() function causes the library to examine the information associated
with an ELF descriptor, elf, and to recalculate the structural data needed to generate
the file’s image.

The cmd argument can have the following values:

ELF_C_NULL       This value tells elf_update() to recalculate various values,
                  updating only the ELF descriptor’s memory structures. Any
                  modified structures are flagged with the ELF_F_DIRTY 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 elf_begin(3ELF)).

ELF_C_WRITE      If cmd has this value, elf_update() duplicates its ELF_C_NULL
                  actions and also writes any “dirty” information associated with
                  the ELF descriptor to the file. That is, when a program has used
                  elf_getdata(3ELF) or the elf_flagdata(3ELF) facilities to
                  supply new (or update existing) information for an ELF descriptor,
                  those data will be examined, coordinated, translated if necessary
                  (see elf32_xlatetof(3ELF)), and written to the file. When
                  portions of the file are written, any ELF_F_DIRTY bits are reset,
                  indicating those items no longer need to be written to the file (see
                  elf_flagdata(3ELF)). 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 elf_begin(), it must have allowed writing the file. That is,
                  the elf_begin() command must have been either ELF_C_RDWR
                  or ELF_C_WRITE.

If elf_update() succeeds, it returns the total size of the file image (not the memory
image), in bytes. Otherwise an error occurred, and the function returns −1.

When updating the internal structures, elf_update() 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>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
</table>

Extended Library Functions 193
elf_update(3ELF)

<table>
<thead>
<tr>
<th>e_ident[EL_DATA]</th>
<th>Library controls other e_ident values</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_type</td>
<td></td>
</tr>
<tr>
<td>e_machine</td>
<td></td>
</tr>
<tr>
<td>e_version</td>
<td></td>
</tr>
<tr>
<td>e_entry</td>
<td></td>
</tr>
<tr>
<td>e_phoff</td>
<td>Only when ELF_P_LAYOUT asserted</td>
</tr>
<tr>
<td>e_shoff</td>
<td>Only when ELF_P_LAYOUT asserted</td>
</tr>
<tr>
<td>e_flags</td>
<td></td>
</tr>
<tr>
<td>e_shstrndx</td>
<td></td>
</tr>
</tbody>
</table>

The following table shows the Program Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_type</td>
<td>The application controls all</td>
</tr>
<tr>
<td>p_offset</td>
<td>program header entries</td>
</tr>
<tr>
<td>p_vaddr</td>
<td></td>
</tr>
<tr>
<td>p_paddr</td>
<td></td>
</tr>
<tr>
<td>p_filesz</td>
<td></td>
</tr>
<tr>
<td>p_memsz</td>
<td></td>
</tr>
<tr>
<td>p_flags</td>
<td></td>
</tr>
<tr>
<td>p_align</td>
<td></td>
</tr>
</tbody>
</table>

The following table shows the Section Header members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh_name</td>
<td></td>
</tr>
<tr>
<td>sh_type</td>
<td></td>
</tr>
<tr>
<td>sh_flags</td>
<td></td>
</tr>
</tbody>
</table>
sh_addr
sh_offset Only when ELF_F_LAYOUT asserted
sh_size Only when ELF_F_LAYOUT asserted
sh_link
sh_info
sh_addralign Only when ELF_F_LAYOUT asserted
sh_entsize

The following table shows the Data Descriptor members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_buf</td>
<td></td>
</tr>
<tr>
<td>d_type</td>
<td></td>
</tr>
<tr>
<td>d_size</td>
<td></td>
</tr>
<tr>
<td>d_off</td>
<td>Only when ELF_F_LAYOUT asserted</td>
</tr>
<tr>
<td>d_align</td>
<td></td>
</tr>
<tr>
<td>d_version</td>
<td></td>
</tr>
</tbody>
</table>

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.
elf_update(3ELF)

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>

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

```
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

unsigned elf_version(unsigned ver);
```

### DESCRIPTION

As elf(3ELF) explains, the program, the library, and an object file have independent notions of the latest ELF version. `elf_version()` lets a program query the ELF library’s internal version. It further lets the program specify what memory types it uses by giving its own working version, `ver`, to the library. Every program that uses the ELF library must coordinate versions as described below.

The header `<libelf.h>` supplies the version to the program with the macro `EV_CURRENT`. 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, `elf_version()` will not accept a working version unknown to the library.

Passing `ver` equal to `EV_NONE` causes `elf_version()` to return the library’s internal version, without altering the working version. If `ver` is a version known to the library, `elf_version()` returns the previous (or initial) working version number. Otherwise, the working version remains unchanged and `elf_version()` returns `EV_NONE`.

### EXAMPLES

**EXAMPLE 1** A sample display of using the `elf_version()` function.

The following excerpt from an application program protects itself from using an older library:

```c
if (elf_version(EV_CURRENT) == EV_NONE) {
    /* library out of date */
    /* recover from error */
}
```

### 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>

### SEE ALSO

elf(3ELF), elf32_xlatetof(3ELF), elf_begin(3ELF), libelf(3LIB), attributes(5)

### 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.
erf(3M)

NAME	
erf, erfc – error and complementary error functions

SYNOPSIS	
c [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double erf(double x);
double erfc(double x);

DESCRIPTION	
The erf() function computes the error function of x, defined as:

\[ \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \]

The erfc() function computes \(1.0 - \text{erf}(x)\).

RETURN VALUES	
Upon successful completion, erf() and erfc() 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 erfc() function is provided because of the extreme loss of relative accuracy if erf(x) is called for large x and the result subtracted from 1.0.

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>

SEE ALSO	
isnan(3M), attributes(5)
exp(3M)

NAME
exp – exponential function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double exp(double x);

DESCRIPTION
The exp() function computes the exponential of x, defined as \( e^x \).

RETURN VALUES
Upon successful completion, exp() returns the exponential of x.

If the correct value would cause overflow, exp() returns HUGE_VAL and sets errno to ERANGE.

If the correct value would cause underflow to zero, exp() returns 0 and may set errno to ERANGE.

If x is NaN, NaN is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The exp() function will fail if:

ERANGE
The result overflows.

The exp() function may fail if:

ERANGE
The result underflows.

USAGE
An application wishing to check for error situations should set errno to 0 before calling exp(). 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>
<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>

SEE ALSO
isnan(3M), log(3M), matherr(3M), mp(3MP), attributes(5), standards(5)

NOTES
Prior to Solaris 2.6, there was a conflict between the pow function in this library and the pow function in the libmp library. This conflict was resolved by prepending mp_ to all functions in the libmp library. See mp(3MP) for details.
expm1(3M)

NAME expm1 – computes exponential functions

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double expm1(double x);

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 \texttt{HUGE_VAL}.

ERRORS No errors will occur.

USAGE The value of \( \text{expm1}(x) \) may be more accurate than \( \text{exp}(x) - 1.0 \) for small values of \( x \).

The \( \text{expm1}() \) and \( \text{log1p}(3M) \) functions are useful for financial calculations of \( ((1+x)^n - 1)/x \), namely:

\[
\text{expm1}(n \cdot \text{log1p}(x)) / x
\]

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>
<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>

SEE ALSO exp(3M), ilogb(3M), log1p(3M), attributes(5)
fabs() function computes the absolute value of $x$, $|x|$. Upon successful completion, fabs() returns the absolute value of $x$. If $x$ is NaN, NaN is returned.

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>

See also isnan(3M), attributes(5)
Your text is already in a natural plain text format. Here is a structured version of the content:

**NAME**
floor – floor function

**SYNOPSIS**
```
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double floor(double x);
```

**DESCRIPTION**
The `floor()` function computes the largest integral value not greater than `x`.

**RETURN VALUES**
Upon successful completion, `floor()` returns the largest integral value not greater than `x`, expressed as a `double`.

- If `x` is NaN, NaN is returned.
- If `x` is ±Inf or ±0, `x` is returned.

**ERRORS**
No errors will occur.

**USAGE**
The integral value returned by `floor()` as a `double` might not be expressible as an `int` or `long int`. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

**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>

**SEE ALSO**
`ceil(3M), isnan(3M), attributes(5)`
fmod(3M)

NAME
fmod – floating-point remainder value function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double fmod(double x, double y);

DESCRIPTION
The fmod() function returns the floating-point remainder of the division of x by y.

RETURN VALUES
The fmod() function returns the value x - i * y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

If x or y is NaN, NaN is returned. If y is 0, NaN is returned and errno is set to EDOM. If x is ±Inf, NaN is returned. If y is non-zero, fmod(±0, y) returns the value of x. If x is not ±Inf, fmod(x, ±Inf) returns the value of x.

ERRORS
The fmod() function may fail if:

EDOM
   y is 0.

No other errors will occur.

USAGE
Portable applications should not call fmod() with y equal to 0, because the result is implementation-dependent. The application should verify y is non-zero before calling fmod().

An application wishing to check for error situations should set errno to 0 before calling fmod(). 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>
<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>

SEE ALSO
isnan(3M), attributes(5)
freeDmiString(3DMI)

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:

<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>

SEE ALSO  
newDmiString(3DMI), libdmi(3LIB), attributes(5)
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_xslatetom – generic class-independent ELF interface

SYNOPSIS
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_relo(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);
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 API's.

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

gelf_checksum()  An analog to elf32_checksum(3ELF) and elf64_checksum(3ELF).
gelf_getclass()  Returns one of the constants ELFCLASS32, ELFCLASS64 or ELFCLASSNONE.
gelf_fsize()  An analog to elf32_fsize(3ELF) and elf64_fsize(3ELF).
gelf_getehdr()  An analog to elf32_getehdr(3ELF) and elf64_getehdr(3ELF). dst points to the location where the GElf_Ehdr header will be stored.
gelf_update_ehdr()  Copies the contents of the GElf_Ehdr ELF header to the underlying Elf32_Ehdr or Elf64_Ehdr structure.
gelf_newehdr()  An analog to elf32_newehdr(3ELF) and elf64_newehdr(3ELF).
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gelf_getphdr()</td>
<td>An analog to elf32_getphdr(3ELF) and elf64_getphdr(3ELF). dst points to the location where the GElf_Phdr program header will be stored.</td>
</tr>
<tr>
<td>gelf_update_phdr()</td>
<td>Copies of the contents of GElf_Phdr program header to underlying the Elf32_Phdr or Elf64_Phdr structure.</td>
</tr>
<tr>
<td>gelf_newphdr()</td>
<td>An analog to elf32_newphdr(3ELF) and elf64_newphdr(3ELF).</td>
</tr>
<tr>
<td>gelf_getshdr()</td>
<td>An analog to elf32_getshdr(3ELF) and elf64_getshdr(3ELF). dst points to the location where the GElf_Shdr section header will be stored.</td>
</tr>
<tr>
<td>gelf_update_shdr()</td>
<td>Copies of the contents of GElf_Shdr section header to underlying the Elf32_Shdr or Elf64_Shdr structure.</td>
</tr>
<tr>
<td>gelf_xlatetof()</td>
<td>An analog to elf32_xlatetof(3ELF) and elf64_xlatetof(3ELF)</td>
</tr>
<tr>
<td>gelf_xlatetom()</td>
<td>An analog to elf32_xlatetom(3ELF) and elf64_xlatetom(3ELF)</td>
</tr>
<tr>
<td>gelf_getsym()</td>
<td>Retrieves the Elf32_Sym or Elf64_Sym information from the symbol table at the given index. dst points to the location where the GElf_Sym symbol entry will be stored.</td>
</tr>
<tr>
<td>gelf_update_sym()</td>
<td>Copies the GElf_Sym information back into the underlying Elf32_Sym or Elf64_Sym structure at the given index.</td>
</tr>
<tr>
<td>gelf_getdyn()</td>
<td>Retrieves the Elf32_Dyn or Elf64_Dyn information from the dynamic table at the given index. dst points to the location where the GElf_Dyn dynamic entry will be stored.</td>
</tr>
<tr>
<td>gelf_update_dyn()</td>
<td>Copies the GElf_Dyn information back into the underlying Elf32_Dyn or Elf64_Dyn structure at the given index.</td>
</tr>
<tr>
<td>gelf_getrela()</td>
<td>Retrieves the Elf32_Rela or Elf64_Rela information from the relocation table at the given index. dst points to the location where the GElf_Rela relocation entry will be stored.</td>
</tr>
<tr>
<td>gelf_update_rela()</td>
<td>Copies the GElf_Rela information back into the underlying Elf32_Rela or Elf64_Rela structure at the given index.</td>
</tr>
</tbody>
</table>
gelf(3ELF)

**gelf_getrel()** Retrieves the Elf32_Rel or Elf64_Rel information from the relocation table at the given index. *dst* points to the location where the GElf_Rel relocation entry will be stored.

**gelf_update_rel()** Copies the GElf_Rel information back into the underlying Elf32_Rel or Elf64_Rel structure at the given index.

**gelf_getsyminfo()** Retrieves the Elf32_Syminfo or Elf64_Syminfo information from the relocation table at the given index. *dst* points to the location where the GElf_Syminfo symbol information entry will be stored.

**gelf_update_syminfo()** Copies the GElf_Syminfo information back into the underlying Elf32_Syminfo or Elf64_Syminfo structure at the given index.

**gelf_getmove()** Retrieves the Elf32_Move or Elf64_Move information from the move table at the given index. *dst* points to the location where the GElf_Move move entry will be stored.

**gelf_update_move()** Copies the GElf_Move information back into the underlying Elf32_Move or Elf64_Move 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

```c
#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) {
        int ii = 0;
        while ((shdr = elf_getshdr(scn, ii)) != NULL) {
            ii++;
        }
        data = elf_newdata(elf, shdr, header.shdrinfo
```

208  man pages section 3: Extended Library Functions • Last Revised 11 July 2001
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:

<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>Safe</td>
</tr>
</tbody>
</table>

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
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);

DESCRIPTION
When first called, getacdir() provides information about the first audit directory in
the audit_control file; thereafter, it returns the next directory in the file. Successive
calls list all the directories listed in audit_control(4) The parameter len specifies the
length of the buffer dir. On return, dir points to the directory entry.

getacmin() reads the minimum value from the audit_control file and returns the
value in min_val. The minimum value specifies how full the file system to which the
audit files are being written can get before the script audit_warn(1M) is invoked.

getacflg() reads the system audit value from the audit_control file and returns
the value in auditstring. The parameter len specifies the length of the buffer auditstring.

getacna() reads the system audit value for non-attributable audit events from the
audit_control file and returns the value in auditstring. The parameter len specifies the
length of the buffer auditstring. Non-attributable events are events that cannot be
attributed to an individual user. inetd(1M) and several other daemons record
non-attributable events.

Calling setac rewind the audit_control file to allow repeated searches.

Calling endac closes the audit_control file when processing is complete.

FILES
/etc/security/audit_control contains default parameters read by the
audit daemon, auditd(1M)

RETURN VALUES
getacdir(), getacflg(), getacna() and getacmin() return:
0 on success.
-2 on failure and set errno to indicate the error.

getacmin() and getacflg() return:
1 on EOF.

getacdir() returns:
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:

-1 on EOF.

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

getcdir(), 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:

<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>

**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.
getauclassent(3BSM)

NAME  
getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r

get audit_class entry

SYNOPSIS  
cce [ flag ... ] file ... -bsm -ls mish -lint1 [ 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);

DESCRIPTION  
getauclassent() and getauclassnam() each return an audit_class entry.

getauclassnam() searches for an audit_class entry with a given class name name.

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
declared in <bsm/libbsm.h> with the following members:

char     *ac_name;
au_class_t     ac_class;
char     *ac_desc;

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.

212  man pages section 3: Extended Library Functions  •  Last Revised 29 Dec 1996
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

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
cc [ flag ... ] file ... -lbsm -lsocket -lnsl -lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>

int getauditflagsbin(char *auditstring, au_mask_t *masks);
int getauditflagschar(char *auditstring, au_mask_t *masks, int verbose);

DESCRIPTION
getauditflagsbin() converts the character representation of audit values pointed to by auditstring into au_mask_t fields pointed to by masks. 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 masks into a string pointed to by auditstring. If verbose is zero, the short (2-character) flag names are used. If verbose is non-zero, the long flag names are used. auditstring should be large enough to contain the ASCII representation of the events.

auditstring contains a series of event names, each one identifying a single audit class, separated by commas. The au_mask_t fields pointed to by masks 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>
<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>

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.
NAME
getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent,
getauevent_r, getauevnam_r, getauevnum_r – get audit_event entry

SYNOPSIS
cce [ 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 *getaevnam(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);

DESCRIPTION
These interfaces document the programming interface for obtaining entries from the
audit_event(4) file. getauevent(), getauevnam(), getauevnum(),
getauevnonam(), 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.
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 `ac_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:

```c
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**

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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

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.
### SYNOPSIS

```c
cc [ flag... ] file... -lsecdb -lsocket -lnsl -lintl [ library... ]
#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);
```

### DESCRIPTION

The `getauthattr()` and `getauthnam()` functions each return an `auth_attr` entry. Entries can come from any of the sources specified in the `nsswitch.conf` file.

The `getauthattr()` function enumerates `auth_attr` entries. The `getauthnam()` function searches for an `auth_attr` entry with a given authorization name `name`. Successive calls to these functions return either successive `auth_attr` entries or `NULL`.

The internal representation of an `auth_attr` entry is an `authattr_t` structure defined in `<auth_attr.h>` with the following members:

```c
cchar name;     /* name of the authorization */
cchar res1;    /* reserved for future use */
cchar res2;    /* reserved for future use */
cchar short_desc; /* short description */
cchar long_desc; /* long description */
kvat_t *attr;  /* array of key-value pair attributes */
```

The `setauthattr()` function “rewinds” to the beginning of the enumeration of `auth_attr` entries. Calls to `getauthnam()` can leave the enumeration in an indeterminate state. Therefore, `setauthattr()` should be called before the first call to `getauthattr()`.

The `endauthattr()` function may be called to indicate that `auth_attr` processing is complete; the system may then close any open `auth_attr` file, deallocate storage, and so forth.

The `chkauthattr()` function verifies whether or not a user has a given authorization. It first reads the `AUTHS_GRANTED` key in the `/etc/security/policy.conf` file and returns 1 if it finds a match for the given authorization. If `chkauthattr()` does not find a match, it reads the `PROFS_GRANTED` key in `/etc/security/policy.conf` and returns 1 if the given authorization is in any profiles specified with the `PROFS_GRANTED` keyword. If a match is not found from the default authorizations and default profiles,
chkauthattr() reads the user_attr(4) database. If it does not find a match in user_attr, it reads the prof_attr(4) database, using the list of profiles assigned to the user, and checks if any of the profiles assigned to the user has the given authorization. The chkauthattr() function returns 0 if it does not find a match in any of the three sources.

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

- The authorization name matches exactly any authorization assigned in the user_attr or prof_attr databases (authorization names are case-sensitive).
- The authorization name suffix is not the 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.

<table>
<thead>
<tr>
<th>Authorization name</th>
<th>/etc/security/policy.conf or user_attr or prof_attr entry</th>
<th>Is user authorized?</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.sun.printer.postscript</td>
<td>com.sun.printer.postscript</td>
<td>Yes</td>
</tr>
<tr>
<td>com.sun.printer.postscript</td>
<td>com.sun.printer.*</td>
<td>Yes</td>
</tr>
<tr>
<td>com.sun.printer.grant</td>
<td>com.sun.printer.*</td>
<td>No</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<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>

SEE ALSO
getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB), auth_attr(4), nsswitch.conf(4), prof_attr(4), user_attr(4), attributes(5), rbac(5)
The `getaudituserent()`, `getauditusernam()`, `getaudituserent_r()`, and `getauditusernam_r()` functions each return an `audit_user` entry. Entries can come from any of the sources specified in the `/etc/nsswitch.conf` file (see `nsswitch.conf(4)`).

The `getauditusernam()` and `getauditusernam_r()` functions search for an `audit_user` entry with a given login name `name`.

The `getaudituserent()` and `getaudituserent_r()` functions enumerate `audit_user` entries; successive calls to these functions will return either successive `audit_user` entries or `NULL`.

The `setaudituser()` function “rewinds” to the beginning of the enumeration of `audit_user` entries. Calls to `getauditusernam()` and `getauditusernam_r()` may leave the enumeration in an indeterminate state, so `setaudituser()` should be called before the first call to `getaudituserent()` or `getaudituserent_r()`.

The `endaudituser()` function may be called to indicate that `audit_user` processing is complete; the system may then close any open `audit_user` file, deallocate storage, and so forth.

The `getaudituserent_r()` and `getauditusernam_r()` functions both take an argument `u`, which is a pointer to an `au_user_ent`. This is the pointer that is returned on successful function calls.

The internal representation of an `audit_user` entry is an `au_user_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```c
struct au_user_ent
{
    char *au_name;
    au_mask_t au_always;
    au_mask_t au_never;
};
```

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

**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**
- `/etc/security/audit_user` stores per-user audit event mask
- `/etc/passwd` stores user-id to username mappings
- `/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>MT-Level</td>
<td>MT-Safe with exceptions.</td>
</tr>
</tbody>
</table>

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

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

The `getauusername()` and `getauuserent()` functions are not MT-safe. The `getauusername_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.
The `getddent()` and `getddnam()` functions each return a `device_deallocate` entry. The `getddent()` function enumerates all `device_deallocate` entries. Successive calls to this function return either successive `device_deallocate` entries or `NULL`. The `getddnam()` function searches for a `device_deallocate` entry with a given device name.

The internal representation of a `device_deallocate` entry is a `devdealloc_t` structure defined in `<bsm/devices.h>` with the following members:

```c
char *dd_devname; /* device allocation name */
char *dd_logout; /* deallocation action on user logout */
char *dd_boot; /* deallocation action on system boot */
```

The `setddent()` function "rewinds" to the beginning of the enumeration of `device_deallocate` entries. Calls to `getddnam()` may leave the enumeration in an indeterminate state, so `setddent()` should be called before the first call to `getddent()`.

The `endddent()` function can be called to indicate that `device_deallocate` processing is complete. The library can then close any open `device_deallocate` file, deallocate any internal storage, and so forth.

The `setddfile()` function changes the pathname used by the other functions for opening the `device_deallocate` file, allowing use of `device_deallocate` files other than the default file, `/etc/security/device_deallocate`.

The `getddent()` function returns a pointer to a `devdealloc_t` if it successfully enumerates an entry. Otherwise it returns `NULL`, indicating the end of the enumeration.

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

**FILES**

/`etc/security/device_deallocate`

Administrative file defining parameters for device deallocation.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:
The `getddent()` and `getddnam()` functions allocate memory for the pointers they return. This memory can be deallocated with the `free(3C)` function.

### 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.
The getdmapent(), getdmapnam(), getdmapdev(), and getdmaptype() functions each return a device_deallocate 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:

```
char *dmap_devname; /* device allocation name */
char *dmap_devtype; /* generic device type */
char *dmap_devlist; /* list of associated device special files */
```

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.

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.
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:

<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>

**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
cc [ flag... ] file... -lsecdb -lsocket -lnsl -lintl [ library... ]
#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);

DESCRIPTION
The getexecattr() function returns a single exec_attr entry. Entries can come
from any of the sources specified in the nsswitch.conf(4) file.

Successive calls to getexecattr() return either successive exec_attr entries or
NULL. Because getexecattr() always returns a single entry, the next pointer in the
execattr_t data structure points to NULL.

The internal representation of an exec_attr entry is an execattr_t structure
defined in <exec_attr.h> with the following members:

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 */

The free_execattr() function releases memory. It follows the next pointers in the
execattr_t structure so that the entire linked list is released.

The setexecattr() function “rewinds” to the beginning of the enumeration of
exec_attr entries. Calls to getexecuser() can leave the enumeration in an
indeterminate state. Therefore, setexecattr() should be called before the first call
to getexecattr().

The endexecattr() function can be called to indicate that exec_attr processing is
complete; the library can then close any open exec_attr file, deallocate any internal
storage, and so forth.
The `getexecuser()` function returns a linked list of entries filtered by the function’s arguments. Only entries assigned to the specified `username`, as described in the `passwd(4)` database, and containing the specified `type` and `id`, as described in the `exec_attr(4)` database, are placed in the list. The `getexecuser()` function is different from the other functions in its family because it spans two databases. It first looks up the list of profiles assigned to a user in the `user_attr` database and the list of default profiles in `/etc/security/policy.conf`, then looks up each profile in the `exec_attr` database.

The `getexecprof()` function returns a linked list of entries that have components matching the function’s arguments. Only entries in the database matching the argument `profname`, as described in `exec_attr`, and containing the `type` and `id`, also described in `exec_attr`, are placed in the list.

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

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

**RETURN VALUES**

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

**USAGE**

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

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

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

EXAMPLES

EXAMPLE 1 The following finds all profiles that have the ping command.
if ((execprof=getexecprof(NULL, KV_COMMAND, "/usr/sbin/ping", GET_ONE)) == NULL) {
    /* do error */
}

EXAMPLE 2 The following finds the entry for the ping command in the Network Administration Profile.
if ((execprof=getexecprof("Network Administration", KV_COMMAND, "/usr/sbin/ping", GET_ALL))==NULL) {
    /* do error */
}

EXAMPLE 3 The following tells everything that can be done in the Filesystem Security profile.
if ((execprof=getexecprof("Filesystem Security", KV_NULL, NULL, GET_ALL))==NULL) {
    /* do error */
}

EXAMPLE 4 The following tells if the tar command is in a profile assigned to user wetmore. If there is no exact profile entry, the wildcard (*), if defined, is returned.
if ((execprof=getexecuser("wetmore", KV_COMMAND, "/usr/bin/tar", GET_ONE)) == NULL) {
    /* do error */
}

FILES

/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:

<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>

SEE ALSO

getauthattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB), exec_attr(4), policy.conf(4), user_attr(4), attributes(5)
NAME
getfauditflags – generates the process audit state

SYNOPSIS
cce [ flag ... ] file ... -lbsm -lsocket -lns1 -lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>

int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks,
    au_mask_t *lastmasks);

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 always 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 never 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
structure 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>
<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>

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  cc [ flag... ] file... -lsedcbl -lsocket -lnscl -lintl [ library... ]

#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);

DESCRIPTION  The getprofattr() and getprofnam() functions each return a prof_attr entry. Entries can come from any of the sources specified in the nsswitch.conf(4) file.

The getprofattr() function enumerates prof_attr entries. The getprofnam() function searches for a prof_attr entry with a given name. Successive calls to these functions return either successive prof_attr entries or NULL.

The internal representation of a prof_attr entry is a profattr_t structure defined in <prof_attr.h> with the following members:

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 */

The free_profattr() function releases memory allocated by the getprofattr() and getprofnam() functions.

The setprofattr() function “rewinds” to the beginning of the enumeration of prof_attr entries. Calls to getprofnam() can leave the enumeration in an indeterminate state. Therefore, setprofattr() should be called before the first call to getprofattr().

The endprofattr() function may be called to indicate that prof_attr processing is complete; the system may then close any open prof_attr file, deallocate storage, and so forth.

The getproflist() function searches for the list of sub-profiles found in the given profname and allocates memory to store this list in proflist. The given profname will be included in the list of sub-profiles. The profcnt argument indicates the number of items currently valid in proflist. Memory allocated by getproflist() should be freed using the free_proflist() function.
The `free_proflist()` function frees memory allocated by the `getproflist()` function. The `profcnt` argument specifies the number of items to free from the `proflist` argument.

**RETURN VALUES**

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

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

**USAGE**

Individual attributes in the `prof_attr_t` structure can be referred to by calling the `kva_match(3SECDB)` 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 `getprofattr()` and `getprofnam()` functions both allocate memory for the pointers they return. This memory should be deallocated with the `free_profattr()` 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.

**FILES**

`/etc/security/prof_attr` profiles and their descriptions

**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>

**SEE ALSO**

`auths(1), profiles(1), getexecattr(3SECDB), getauthattr(3SECDB), prof_attr(4)`
**NAME**
getprojent, getprojbyname, getprojbyid, getdefaultproj, inproj, getprojidbyname, setprojent, endprojent, fgetprojent – project database entry functions

**SYNOPSIS**
cc [ flag... ] file... -lproject [ library... ]
#include <project.h>

```c
struct project *getprojent(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 setprojent(void);
void endprojent(void);
struct project *fgetprojent(FILE *, struct project *proj, void *buffer, size_t bufsize);
```

**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 /etc/nsswitch.conf file (see nsswitch.conf(4)).

The setprojent(), getprojent(), and endprojent() functions are used to enumerate project entries from the database.

The setprojent() 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 getprojent().

The getprojent() function returns a pointer to a structure containing the broken-out fields of an entry in the project database. When first called, getprojent() 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 endprojent() 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 endprojent().

The getprojbyname() function searches the project database for an entry with the project name specified by the character string name.
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
enumerate disjoint subsets of the project database. The inproj(),
getprojbyname(), getprojbyid(), and getdefaultproj() functions leave the
enumeration position in an indeterminate state.

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

```c
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 getprojent() 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 getprojent() 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 getprojent(), getprojbyname(), getprojbyid(), inproj(),
getprojidbyname(), fgetprojent(), and getdefaultproj() functions will fail if:

- **EINTR** A signal was caught during the operation.
- **EIO** An I/O error has occurred.
- **EMFILE** There are OPEN_MAX 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 buffer and bufsize to contain
  the data to be referenced by the resulting project structure.
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 getprojent() 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:

<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>See “Reentrant Interfaces” in Description</td>
</tr>
</tbody>
</table>

**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  cc [ flag... ] file... -lssecdb -lssocket -lns1 -lintl [ 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);

DESCRIPTION  The getuserattr(), getusernam(), and getuseruid() functions each return a user_attr(4) entry. Entries can come from any of the sources specified in the nsswitch.conf(4) file. The getuserattr() function enumerates user_attr entries. The getusernam() function searches for a user_attr entry with a given user name name. The getuseruid() function searches for a user_attr entry with a given user id uid. Successive calls to these functions return either successive user_attr entries or NULL.

The free_userattr() function releases memory allocated by the getusernam() and getuserattr() functions.

The internal representation of a user_attr entry is a userattr_t structure defined in <user_attr.h> with the following members:

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 */

The setuserattr() function “rewinds” to the beginning of the enumeration of user_attr entries. Calls to getusernam() may leave the enumeration in an indeterminate state, so setuserattr() should be called before the first call to getuserattr().

The enduserattr() function may be called to indicate that user_attr processing is complete; the library may then close any open user_attr file, deallocate any internal storage, and so forth.

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

The getusernam() function returns a pointer to a userattr_t if it successfully locates the requested entry; otherwise it returns NULL.
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.

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

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

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

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

<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>

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

Extended Library Functions

237
NAME  
gmatch – shell global pattern matching

SYNOPSIS  
cc [ flag ... ] file ... -lgen [ library ... ]  
#include <libgen.h>  
    int gmatch(const char *str, const char *pattern);

DESCRIPTION  
gmatch() checks whether the null-terminated string str matches the null-terminated pattern string pattern. 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.

    char *s;
    gmatch(s, "*[a-]" )

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>

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.
The `hypot()` function computes the length of the hypotenuse of a right-angled triangle:

\[ \sqrt{x^2 + y^2} \]

**RETURN VALUES**

Upon successful completion, the `hypot()` function returns the length of the hypotenuse of a right-angled triangle with sides of length `x` and `y`.

If the result would cause overflow, `HUGE_VAL` is returned and `errno` may be set to `ERANGE`.

If `x` or `y` is NaN, NaN is returned.

**ERRORS**

The `hypot()` function may fail:

- **ERANGE** The result overflows.

**USAGE**

The `hypot()` function takes precautions against underflow and overflow during intermediate steps of the computation.

An application wishing to check for error situations should set `errno` to 0 before calling `hypot()`. If `errno` is non-zero on return, or the return value is `HUGE_VAL` or NaN, 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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`isnan(3M), sqrt(3M), attributes(5)`
ilogb(3M)

NAME
ilogb – returns an unbiased exponent

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

int ilogb(double x);

DESCRIPTION
The ilogb() 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, ilogb() returns the exponent part of \( x \).

If \( x \) is 0, ilogb() returns \(-\text{INT\_MAX}\).

If \( x \) is NaN or ±Inf, ilogb() returns \text{INT\_MAX}.

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>

SEE ALSO
ilogb(3M), attributes(5)
isencrypt – determine whether a buffer of characters is encrypted

#include<libgen.h>

int isencrypt(const char *fbuf, size_t ninbuf);

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 ninbuf 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 ninbuf 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.

If the buffer is encrypted, 1 is returned; otherwise, zero is returned.

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>

setlocale(3C), attributes(5)

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:

<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>

SEE ALSO    attributes(5)
j0, j1, jn – Bessel functions of the first kind

SYNOPSIS

cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>
double j0(double x);
double j1(double x);
double jn(int n, double x);

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 errno may be set to ERANGE.

If x is NaN, NaN is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS

The j0(), j1() and jn() functions may fail if:

ERANGE The value of x was too large in magnitude.

USAGE

An application wishing to check for error situations should set errno to 0 before calling j0(), j1() or jn(). 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>
<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>

SEE ALSO

isnan(3M), matherr(3M), y0(3M), attributes(5), standards(5)
kstat – Perl tied hash interface to the kstat facility

use Sun::Solaris::Kstat;
Sun::Solaris::Kstat->new();
Sun::Solaris::Kstat->update();
Sun::Solaris::Kstat->{module}{instance}{name}{statistic}

Kernel statistics are categorized using a 3-part key consisting of the module, the instance, and the statistic name. For example, CPU information can be found under cpu_stat:0:cpu_stat0, as in the above example. The method Sun::Solaris::Kstat->new() creates a new 3-layer tree of Perl hashes with the same structure; that is, the statistic for CPU 0 can be accessed as $ks->{cpu_stat}{0}{cpu_stat0}. The fourth and lowest layer is a tied hash used to hold the individual statistics values for a particular system resource.

For performance reasons, the creation of a Sun::Solaris::Kstat object is not accompanied by a following read of all possible statistics. Instead, the 3-layer structure described above is created, but reads of a statistic's values are done only when referenced. For example, accessing $ks->{cpu_stat}{0}{cpu_stat0}{syscall} will read in all the statistics for CPU 0, including user, system, and wait times, and the other CPU statistics, as well as the number of system call entries. Once you have accessed a lowest level statistics value, calling $ks->update will automatically update all the individual values of any statistics you have accessed.

Note that there are two values of the lowest-level hash that can be read without causing the full set of statistics to be read from the kernel. These are "class", which is the kstat class of the statistics, and "crtime", which is the time that the kstat was created. See kstat(3KSTAT) for full details of these fields.

Methods

new() Create a new kstat statistics hierarchy and return a reference to the top-level hash. Use it like any normal hash to access the statistics.

update() Update all the statistics that have been accessed so far. In scalar context, update() returns 1 if the kstat structure has changed, and 0 otherwise. In list context, update() returns references to two arrays: the first holds the keys of any kstats that have been added, and the second holds the keys of any kstats that have been deleted. Each key will be returned in the form "module:instance:name".

EXAMPLES

EXAMPLE 1 Sun::Solaris::Kstat example
use Sun::Solaris::Kstat;

my $kstat = Sun::Solaris::Kstat->new();
my ($usr1, $sys1, $wio1, $idle1) = @$kstat->{cpu_stat}{0}{cpu_stat0}{syscall}
print("usr sys wio idle\n");
while (1) {
    sleep 5;
    if ($kstat->update() ) {

EXAMPLE 1 Sun::Solaris::Kstat example (Continued)

```perl
        print("Configuration changed\n");
    }
    my ($usr2, $sys2, $wio2, $idle2) =
        @{$kstat->{cpu_stat}{0}{cpu_stat0}}{qw(user kernel wait idle)};
    printf(" %.2d %.2d %.2d %.2d\n",
        ($usr2 - $usr1) / 5, ($sys2 - $sys1) / 5,
        ($wio2 - $wio1) / 5, ($idle2 - $idle1) / 5);
    $usr1 = $usr2;
    $sys1 = $sys2;
    $wio1 = $wio2;
    $idle1 = $idle2;
}
```

SEE ALSO
perl(1), kstat(1M), kstat(3KSTAT), kstat_chain_update(3KSTAT),
kstat_close(3KSTAT), kstat_open(3KSTAT), kstat_read(3KSTAT)

NOTES
As the statistics are stored in a tied hash, taking additional references of members of
the hash, such as
```
my $ref = \$ks->{cpu_stat}{0}{cpu_stat0}{syscall};
print("$$ref\n");
```
will be recorded as a hold on that statistic’s value, preventing it from being updated
by refresh(). Copy the values explicitly if persistence is necessary.

Several of the statistics provided by the kstat facility are stored as 64-bit integer
values. Perl 5 does not yet internally support 64-bit integers, so these values are
approximated in this module. There are two classes of 64-bit value to be dealt with:

64-bit intervals and times
These are the crtime and snaptime fields of all the
statistics hashes, and the wtime, wlen, time,
wlastupdate, rltime, rlen, and rlupdate
fields of the kstat I/O statistics structures. These are
measured by the kstat facility in nanoseconds,
meaning that a 32-bit value would represent
approximately 4 seconds. The alternative is to store the
values as floating-point numbers, which offer
approximately 53 bits of precision on present hardware.
64-bit intervals and timers as floating point values
expressed in seconds, meaning that time-related kstats
are being rounded to approximately microsecond
resolution.

64-bit counters
It is not useful to store these values as 32-bit values. As
noted above, floating-point values offer 53 bits of
precision. Accordingly, all 64-bit counters are stored as
floating-point values.

Extended Library Functions 245
The kstat facility is a general-purpose mechanism for providing kernel statistics to users. The kernel maintains a linked list of statistics structures, or kstats. Each kstat has a common header section and a type-specific data section. The header section is defined by the kstat_t structure:

```c
typedef int kid_t; /* unique kstat id */

typedef struct kstat {
    /*
     * Fields relevant to both kernel and user
     */
    hrtime_t ks_crtime; /* 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;
```

The fields that are of significance to the user are:

- **ks_crtime**: The time the kstat was created. This allows you to compute the rates of various counters since the kstat was created; "rate since boot" is replaced by the more general concept of "rate since kstat creation". All times associated with kstats (such as creation time, last snapshot time, kstat_timer_t and kstat_io_t timestamps, and the like) are 64-bit nanosecond values. The accuracy of kstat timestamps is machine dependent, but the precision (units) is the same across all platforms. See `gethrtime(3C)` for general information about high-resolution timestamps.

- **ks_next**: Kstats are stored as a linked list, or chain. `ks_next` points to the next kstat in the chain.
ks_kid A unique identifier for the kstat.

ks_module, ks_instance contain the name and instance of the module that created the kstat. In cases where there can only be one instance, ks_instance is 0.

ks_name gives a meaningful name to a kstat. The full kstat namespace is <ks_module, ks_instance, ks_name>, so the name only need be unique within a module.

ks_type The type of data in this kstat. Kstat data types are discussed below.

ks_class 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: disk, tape, controller, net, rpc, vm, kvm, hat, streams, kmem, kmem_cache, kstat, and misc. (The kstat class encompasses things like kstat_types.)

ks_data, ksndata, ks_data_size ks_data is a pointer to the kstat's data section. The type of data stored there depends on ks_type. ksndata indicates the number of data records. Only some kstat types support multiple data records. Currently, KSTAT_TYPE_RAW, KSTAT_TYPE_NAMED and KSTAT_TYPE_TIMER kstats support multiple data records. KSTAT_TYPE_INTR and KSTAT_TYPE_IO kstats support only one data record. ks_data_size is the total size of the data section, in bytes.

ks_snaptime The timestamp for the last data snapshot. This allows you to compute activity rates:

rate = (new_count - old_count) / (new_snaptime - old_snaptime);

kstat data types The following types of kstats are currently available:

#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 */

To get a list of all kstat types currently supported in the system, tools can read out the standard system kstat kstat_types (full name spec is "<unix", 0, "kstat_types">). This is a KSTAT_TYPE_NAMED kstat in which the name field describes the type of kstat, and the value field is the kstat type number (for example, KSTAT_TYPE_IO is type 3 -- see above).

Raw kstat KSTAT_TYPE_RAW raw data
The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like `sysinfo`.

### KSTAT_TYPE_NAMED
A list of arbitrary `name=value` statistics.

```c
typedef struct kstat_named {
    char name[KSTAT_STRLEN]; /* name of counter */
    uchar_t data_type; /* data type */
    union {
        char c[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;
    } value; /* value of counter */
} kstat_named_t;
```

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.

```c
#define KSTAT_NAMED_STR_PTR(knptr) ((knptr)->value.string.addr.ptr)
#define KSTAT_NAMED_STR_BUFLEN(knptr) ((knptr)->value.string.len)
```
KSTAT_NAMED_STR_BUFLEN() returns the number of bytes required to store the string pointed to by KSTAT_NAMED_STR_PTR(); that is, 
\texttt{strlen(KSTAT_NAMED_STR_PTR()) + 1}.

**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).

```c
#define KSTAT_INTR_HARD 0
#define KSTAT_INTR_SOFT 1
#define KSTAT_INTR_WATCHDOG 2
#define KSTAT_INTR_SPURIOUS 3
#define KSTAT_INTR_MULTSVC 4
#define KSTAT_NUM_INTRS 5
```

```c
typedef struct kstat_intr {
    uint_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */
} kstat_intr_t;
```

**Event timer kstat**

KSTAT_TYPE_TIMER Event timer statistics.

These provide basic counting and timing information for any type of event.

```c
typedef struct kstat_timer {
    char name[KSTAT_STRLEN]; /* event name */
    uchar_t resv; /* reserved */
    u_longlong_t num_events; /* number of events */
    hrt ime_t elapsed_time; /* cumulative elapsed time */
    hrt ime_t min_time; /* shortest event duration */
    hrt ime_t max_time; /* longest event duration */
    hrt ime_t start_time; /* previous event start time */
    hrt ime_t stop_time; /* previous event stop time */
} kstat_timer_t;
```

**I/O kstat**

KSTAT_TYPE_IO I/O statistics.

```c
typedef struct kstat_io {
    /*
    * Basic counters.
    */
    u_longlong_t read; /* number of bytes read */
    u_longlong_t written; /* number of bytes written */
    uint_t reads; /* number of read operations */
    uint_t writes; /* number of write operations */
    /*
    * Accumulated time and queue length statistics.
    */
} kstat_io_t;
```
* 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.

\[
\begin{array}{c|c|c|c|c}
\hline
\text{Time} & t_1 & t_2 & t_3 & t_4 \\
\hline
\end{array}
\]

* 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.

\[
\text{hrtime_t wtime; /* cumulative wait (pre-service) time */}
\text{hrtime_t wlentime; /* cumulative wait length*time product*/}
\text{hrtime_t wlastupdate; /* last time wait queue changed */}
\text{hrtime_t rtime; /* cumulative run (service) time */}
\text{hrtime_t rlentime; /* cumulative run length*time product */}
\]
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)
The kstat_chain_update() 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 (kstat_t’s) pointed to by kc->kc_chain, which is initialized by kstat_open(3KSTAT). 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 kstat_chain_update() 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, kc->kc_chain_id. If the KCIDs match, kstat_chain_update() does nothing. Otherwise, it deletes any invalid kstat headers from the user’s kstat chain, adds any new ones, and sets kc->kc_chain_id to the new KCID. All other kstat headers in the user’s kstat chain are unmodified.

The kstat_chain_update() function returns the new KCID if the kstat chain has changed, 0 if it hasn’t, or -1 on 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>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
kstat(3KSTAT), kstat_close(3KSTAT), kstat_data_lookup(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT), kstat_read(3KSTAT), kstat_write(3KSTAT), attributes(5)
kstat_lookup(3KSTAT)

NAME

kstat_lookup, kstat_data_lookup – find a kstat by name

SYNOPSIS

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);

DESCRIPTION

The kstat_lookup() function traverses the kstat chain, kc->kc_chain, searching for a kstat with the same ks_module, ks_instance, and ks_name fields; this triplet uniquely identifies a kstat. If ks_module is NULL, ks_instance is -1, or ks_name is NULL, then those fields will be ignored in the search. For example, kstat_lookup(kc, NULL, -1, "foo") will simply find the first kstat with name "foo".

The kstat_data_lookup() function searches the kstat’s data section for the record with the specified name. This operation is valid only for kstat types which have named data records. Currently, only the KSTAT_TYPE_NAMED and KSTAT_TYPE_TIMER kstats have named data records.

RETURN VALUES

The kstat_lookup() function returns a pointer to the requested kstat if it is found, or NULL if it is not.

The kstat_data_lookup() 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, kstat_data_lookup() returns NULL.

FILES

/dev/kstat kernel statistics driver

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_close(3KSTAT), kstat_open(3KSTAT), kstat_read(3KSTAT), kstat_write(3KSTAT), attributes(5)
kstat_open(3KSTAT)

NAME  | kstat_open, kstat_close – initialize kernel statistics facility

SYNOPSIS  | cc[ flag ...] file ... -lkstat [ library ...]
#include <kstat.h>

   kstat_ctl_t *kstat_open(void);
   int kstat_close(kstat_ctl_t *kc);

DESCRIPTION  | The kstat_open() 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 kc argument in subsequent libkstat function calls.

            | The kstat_close() function frees all resources that were associated with kc. This is
            | done automatically on exit(2) and execve() (see exec(2)).

RETURN VALUES  | The kstat_open() function returns a pointer to a kstat control structure. On failure,
              | it returns NULL and no resources are allocated.

            | The kstat_close() function returns 0 on success and -1 on failure.

FILES  | /dev/kstat  kernel statistics driver

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  | kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_data_lookup(3KSTAT),
           | kstat_lookup(3KSTAT), kstat_read(3KSTAT), kstat_write(3KSTAT),
           | attributes(5)
NAME  kstat_read, kstat_write – read or write kstat data

SYNOPSIS  
```c
#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);
```

DESCRIPTION  The `kstat_read()` function gets data from the kernel for the kstat pointed to by `ksp`. `ksp->ks_data` is automatically allocated (or reallocated) to be large enough to hold all of the data. `ksp->ks_nodata` is set to the number of data fields, `ksp->ks_data_size` is set to the total size of the data, and `ksp->ks_snaptime` is set to the high-resolution time at which the data snapshot was taken. If `buf` is non-NULL, the data is copied from `ksp->ks_data` into `buf`.

The `kstat_write()` function writes data from `buf`, or from `ksp->ks_data` if `buf` is NULL, to the corresponding kstat in the kernel. Only the superuser can use `kstat_write()`.

RETURN VALUES  On success, `kstat_read()` and `kstat_write()` return the current kstat chain ID (KCID). On failure, they return -1.

FILES  /dev/kstat kernel statistics driver

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_close(3KSTAT), kstat_data_lookup(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT), attributes(5)
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`. Upon successful completion, the function returns a pointer to the value sought. Otherwise, it returns `NULL`.

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.
The **kvm_getu()** function reads the u-area of the process specified by **proc** to an area of static storage associated with **kd** and returns a pointer to it. Subsequent calls to **kvm_getu()** will overwrite this static area.

The **kd** argument is a pointer to a kernel descriptor returned by **kvm_open(3KVM)**. The **proc** argument is a pointer to a copy in the current process' address space of a **proc** structure, obtained, for instance, by a prior **kvm_nextproc(3KVM)** call.

The **kvm_getcmd()** function constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by **proc**.

The **kd** argument is a pointer to a kernel descriptor returned by **kvm_open(3KVM)**. The **u** argument is a pointer to a copy in the current process' address space of a **user** structure, obtained, for instance, by a prior **kvm_getu()** call. If **arg** is not **NULL**, the command line arguments are formed into a null-terminated array of string pointers. The address of the first such pointer is returned in **arg**. If **env** is not **NULL**, then the environment is formed into a null-terminated array of string pointers. The address of the first of these is returned in **env**.

The pointers returned in **arg** and **env** refer to data allocated by **malloc(3C)** and should be freed by a call to **free()** when no longer needed. See **malloc(3C)** 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, **kvm_getcmd()** will make the best attempt possible, returning \(-1\) if the user process data is unrecognizable.

**RETURN VALUES**

On success, **kvm_getu()** returns a pointer to a copy of the u-area of the process specified by **proc**. On failure, it returns **NULL**.

The **kvm_getcmd()** function returns **0** on success and **−1** on failure.

**ATTRIBUTES**

See **attributes(5)** for descriptions of the following attributes:
kvm_getu(3KVM)

<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>

### 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)`. 
kvm_nextproc, kvm_getproc, kvm_setproc – read system process structures

NAME
kvm_nextproc, kvm_getproc, kvm_setproc

SYNOPSIS
#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);

kvm_nextproc() The kvm_nextproc() function may be used to sequentially read all of the system process structures from the kernel identified by kd (see kvm_open(3KVM)). Each call to kvm_nextproc() 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 kvm_nextproc(), kvm_setproc(), or kvm_getproc(). 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 kvm_nextproc(), and since the cache may contain obsolete information, there is no guarantee that every process structure returned refers to an active process, nor is it certain that all processes will be reported.

kvm_setproc() The kvm_setproc() function rewinds the process list, enabling kvm_nextproc() 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 kvm_getproc() function locates the proc structure of the process specified by pid and returns a pointer to it. This function does not interact with the process table pointer manipulated by kvm_nextproc(); however, the restrictions regarding the validity of the data still apply.

RETURN VALUES
On success, kvm_nextproc() returns a pointer to a copy of the next valid process table entry. On failure, it returns NULL.

On success, kvm_getproc() returns a pointer to the proc structure of the process specified by pid. On failure, it returns NULL.

The kvm_setproc() function returns 0 on success –1 on 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>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

Extended Library Functions 259
kvm_nextproc(3KVM)

SEE ALSO  kvm_getu(3KVM), kvm_open(3KVM), kvm_read(3KVM), attributes(5)
NAME
kvm_nlist – get entries from kernel symbol table

SYNOPSIS
#include <kvm.h>
#include <nlist.h>

int kvm_nlist(kvm_t *kd, struct nlist *nl);

DESCRIPTION
kvm_nlist() examines the symbol table from the kernel image identified by kd (see
kvm_open(3KVM)) and selectively extracts a list of values and puts them in the array
of nlist structures pointed to by nl. The name list pointed to by nl consists of an
array of structures containing names, types and values. The n_name 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
n_name field. For each entry in nl, if the named symbol is present in the kernel symbol
table, its value and type are placed in the n_value and n_type fields. If a symbol cannot
be located, the corresponding n_type field of nl is set to zero.

RETURN VALUES
kvm_nlist() returns the value of nlist(3UCB) or nlist(3ELF), depending on the
library used.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
nlist(3UCB), nlist(3ELF), kvm_open(3KVM), kvm_read(3KVM), attributes(5)
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);

The kvm_open() 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 kd argument in subsequent kernel VM function calls.

The namelist argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in corefile. If namelist is NULL, 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 /dev/ksyms as a default namelist file.

The corefile argument specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see savecore(1M)) or the special device /dev/mem. If corefile is NULL, the currently running kernel is accessed, using /dev/mem and /dev/kmem.

The swapfile argument specifies a file that represents the swap device. If both corefile and swapfile are NULL, the swap device of the currently running kernel is accessed. Otherwise, if swapfile is NULL, kvm_open() may succeed but subsequent kvm_getu(3KVM) function calls may fail if the desired information is swapped out.

The flag function is used to specify read or write access for corefile and may have one of the following values:

O_RDONLY open for reading
O_RDWR open for reading and writing

The errstr 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 kvm_open. Errors are printed to stderr. A useful value to supply for errstr would be argv[0]. This has the effect of printing the process name in front of any error messages.

Applications using libkvm are dependent on the underlying data model of the kernel image, that is, whether it is a 32-bit or 64-bit kernel.

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 libkvm interfaces will fail to open a 64-bit kernel image. Similarly, a 64-bit application that uses the 64-bit version of the libkvm interfaces will fail to open a 32-bit kernel image.
**kvm_close()**

The `kvm_close()` function closes all file descriptors that were associated with `kd`. These files are also closed on `exit(2)` and `execve()` (see `exec(2)`). `kvm_close()` also resets the `proc` pointer associated with `kvm_nextproc(3KVM)` and flushes any cached kernel data.

**RETURN VALUES**

The `kvm_open()` function returns a non-null value suitable for use with subsequent kernel VM function calls. On failure, it returns `NULL` and no files are opened.

The `kvm_close()` function returns 0 on success, -1 on failure.

**FILES**

`/dev/kmem`

`/dev/ksyms`

`/dev/mem`

**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>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`savecore(1M)`, `exec(2)`, `exit(2)`, `pathconf(2)`, `getloadavg(3C)`, `kstat(3KSTAT)`, `kvm_getu(3KVM)`, `kvm_nextproc(3KVM)`, `kvm_nlist(3KVM)`, `kvm_read(3KVM)`, `sysconf(3C)`, `libkvm(3LIB)`, `proc(4)`, `attributes(5)`, `lfcompile(5)``

**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 `libkvm` are likely to be platform- and release-dependent.

On 32-bit systems, applications that use `libkvm` 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 `libkvm` interfaces to access the running kernel must themselves be 64-bit applications.

Most of the traditional uses of `libkvm` 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 `sysconf(3C)`, `proc(4)`, `kstat(3KSTAT)`, `getloadavg(3C)`, and `pathconf(2)`.
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
#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);

kvm_kread() The kvm_kread() function transfers data from the kernel address space to the
address space of the process. nbytes bytes of data are copied from the kernel virtual
address given by addr to the buffer pointed to by buf.

kvm_kwrite() The kvm_kwrite() function is like kvm_kread(), except that the direction of the
transfer is reversed. To use this function, the kvm_open(3KVM) call that returned kd
must have specified write access.

kvm_uread() The kvm_uread() function transfers data from the address space of the processes
specified in the most recent kvm_getu(3KVM) call. nbytes bytes of data are copied
from the user virtual address given by addr to the buffer pointed to by buf.

kvm_uwrite() The kvm_uwrite() function is like kvm_uread(), except that the direction of the
transfer is reversed. To use this function, the kvm_open(3KVM) call that returned kd
must have specified write access. The address is resolved in the address space of
the process specified in the most recent kvm_getu(3KVM) call.

kvm_read() The kvm_read() function transfers data from the kernel image specified by kd (see
kvm_open(3KVM)) to the address space of the process. nbytes bytes of data are copied
from the kernel virtual address given by addr to the buffer pointed to by buf.

kvm_write() The kvm_write() function is like kvm_read(), except that the direction of data
transfer is reversed. To use this function, the kvm_open(3KVM) call that returned kd
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 kvm_getu(3KVM) call.
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:

<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>

**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  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);

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 Linker and Libraries Guide for a full description of the
   link-editor support mechanism.

SEE ALSO  ld(1)

   Linker and Libraries Guide
The `lgamma()`, `gamma()`, `lgamma_r()` and `gamma_r()` 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 `lgamma()` and `gamma()` functions use the external integer `signgam` to return the sign of \( |\Gamma(x)| \) while `lgamma_r()` and `gamma_r()` use the user-allocated space addressed by `signgamp`.

In the case of `lgamma()`, do not use the expression `signgam*exp(lgamma(x))` to compute

\[ g := \Gamma(x) \]

Instead compute `lgamma()` first:

\[ lg = \text{lgamma}(x); \quad g = \text{signgam}*\exp(lg); \]
only after `lgamma()` has returned can `signgam` be correct. Note that \(-\infty\) 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>See NOTES below.</td>
</tr>
</tbody>
</table>

**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)`). It 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.
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 `di_node_t` in the libdevinfo interfaces. Each `di_node_t` 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 `di_binding_name(3DEVINFO)` man page. Each device node also has a physical path, which is accessed by calling `di_devfs_path(3DEVINFO)`.

Properties provide device specific information for device configuration and usage. Properties may be defined by software (`di_prop_t`) or by firmware (`di_prom_prop_t`). One way to access each `di_prop_t` is to make successive calls to `di_prop_next(3DEVINFO)` until `DI_PROP_NIL` is returned. For each `di_prop_t`, use interfaces on the `di_prop_bytes(3DEVINFO)` man page to obtain property names and values. Another way to access these properties is to call `di_prop_lookup_bytes(3DEVINFO)` to find the value of a property with a given name. Accessing a `di_prom_prop_t` is similar to accessing a `di_prop_t`, except that the interface names start with `di_prom_prop` and additional calls to `di_prom_init(3DEVINFO)` and `di_prom_fini(3DEVINFO)` 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 (`di_minor_t`) may be obtained by making successive calls to `di_minor_next(3DEVINFO)` until `DI_MINOR_NIL` is returned. For each minor node, `di_minor_dev(3DEVINFO)` 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/rdsk/c0t2d0s2)
**EXAMPLE 1** Information Accessible Through `libdevinfo` Interfaces  

(Continued)

```
name: c, raw
dev_t: 0x0080012 (32/18)
spectype: IPSCHR (character special)
```

**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>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**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*
libnvpair(3NVPAIR)

NAME
libnvpair – library of name-value pair functions

SYNOPSIS
cc [flag ...] file ...-lnvpair [library ...]
#include <libnvpair.h>

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>
<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>

SEE ALSO
libnvpair(3LIB), attributes(5)
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 `<picl.h>` 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 `picl`, which defines a basic set of properties that all nodes in the tree must possess. The following table shows the property set of a `picl` class node.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node</td>
</tr>
<tr>
<td>_class</td>
<td>The PICL class name of the node</td>
</tr>
<tr>
<td>_parent</td>
<td>Node handle of the parent node</td>
</tr>
<tr>
<td>_child</td>
<td>Node handle of the first child node</td>
</tr>
<tr>
<td>_peer</td>
<td>Node handle of the next peer node</td>
</tr>
</tbody>
</table>

Property names with a leading underscore (`_`) are reserved for use by the PICL framework. The property names `_class`, `_parent`, `_child`, and `_peer` 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 `table` property type has the handle to a table as its value. A table is a matrix of properties. A `reference` property type has a handle to a node in the tree as its value. A `reference` property may be used to establish an association between any two nodes in the tree. A `timestamp` property type has the value of time in seconds since Epoch. A `bytearray` property type has an array of bytes as its value. A `charstring` property type has a null (`\0`) terminated sequence of ASCII characters. The size of a property specifies the size of its value in bytes. A `void` property type denotes a property that exists but has no value.

The following table lists the different PICL property types enumerated in `picl_prop_type_t`. 
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.

```c
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:

```c
#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`.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICL_PTYPE_VOID</td>
<td>None</td>
</tr>
<tr>
<td>PICL_PTYPE_INT</td>
<td>Is an integer</td>
</tr>
<tr>
<td>PICL_PTYPE_UNSIGNED_INT</td>
<td>Is an unsigned integer</td>
</tr>
<tr>
<td>PICL_PTYPE_FLOAT</td>
<td>Is a floating-point number</td>
</tr>
<tr>
<td>PICL_PTYPE_REFERENCE</td>
<td>Is a PICL node handle</td>
</tr>
</tbody>
</table>
libpicl(3PCL)

<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>

SEE ALSO libpicl(3LIB), attributes(5)
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
ptree_propinfo_t structure defined as:

```c
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;
```

See `libpicl(3PICL)` for more information on PICL tree nodes and properties.

The maximum size of a property value cannot exceed `PICL_PROPSIZE_MAX`. It is
currently set to 512KB.

### Volatile Properties

In addition to `PICL_READ` and `PICL_WRITE` property access modes, the plug-in
modules specify whether a property is volatile or not by setting the bit
`PICL_VOLATILE`.

```c
#define PICL_VOLATILE 0x4
```

For a volatile property, the plug-in module provides the access functions to read
and/or write the property in the `ptree_propinfo_t` 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 `ptree_rarg_t`, 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.

```c
typedef struct {
    picl_nodehdl_t nodeh;
    picl_prophdl_t proph;
    door_cred_t cred;
} ptree_rarg_t;
```

The prototype of the read access function for volatile property is:
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/
/usr/platform/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)

<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>

SEE ALSO  
libpicl(3PICL), libpicltree(3LIB), picld_log(3PICLTREE), picld_plugin_register(3PICLTREE), attributes(5)
NAME
libtnfctl – library for TNF probe control in a process or the kernel

SYNOPSIS
cc [ flag ... ] file ... -lt nfctl [ library ... ]
#include <tnf/tnfctl.h>

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:

- **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.

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>
<thead>
<tr>
<th>Interface</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
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
The `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 `tnfctlprobeapply()` and `tnfctlprobeapply_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 `tnfctlprobeenable()` and `tnfctlprobedisable()` functions control
whether the probe, when hit, will be ignored.

The `tnfctlprobe_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 `tnfctlprobe_connect()` and `tnfctlprobe_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 `tnfctl_buffer_dealloc()` function deallocates the kernel’s internal trace buffer.

Upon successful completion, these functions return `TNFCTL_ERR_NONE`.

The error codes for `libtnfctl` are:

- `TNFCTL_ERR_ACCES`: Permission denied.
- `TNFCTL_ERR_NOTARGET`: The target process completed.
- `TNFCTL_ERR_ALLOCFAIL`: A memory allocation failure occurred.
- `TNFCTL_ERR_INTERNAL`: An internal error occurred.
- `TNFCTL_ERR_SIZETOOSMALL`: The requested trace size is too small.
- `TNFCTL_ERR_SIZETOOBIG`: The requested trace size is too big.
- `TNFCTL_ERR_BADARG`: Bad input argument.
- `TNFCTL_ERR_NOTDYNAMIC`: The target is not a dynamic executable.
- `TNFCTL_ERR_NOLIBTNFPROBE`: `libtnfprobe.so` not linked in target.
- `TNFCTL_ERR_BUFBROKEN`: Tracing is broken in the target.
- `TNFCTL_ERR_BUFEXISTS`: A buffer already exists.
- `TNFCTL_ERR_NOBUF`: No buffer exists.
- `TNFCTL_ERR_BADDEALLOC`: Cannot deallocate buffer.
- `TNFCTL_ERR_NOPROCESS`: No such target process exists.
- `TNFCTL_ERR_Filenotfound`: File not found.
- `TNFCTL_ERR_BUSY`: Cannot attach to process or kernel because it is already tracing.
- `TNFCTL_ERR_INVALIDPROBE`: Probe no longer valid.
- `TNFCTL_ERR_USR1`: Error code reserved for user.
- `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.

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>SUNWtnfctl</td>
</tr>
</tbody>
</table>
**libtnfctl(3TNF)**

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

**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_state_get(3TNF),
tnfctl_trace_state_set(3TNF), libtnfctl(3LIB), proc(4), attributes(5)

**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
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double log10(double x);

DESCRIPTION
The log10() function computes the base 10 logarithm of \( x \), \( \log_{10}(x) \). The value of \( x \) must be positive.

RETURN VALUES
Upon successful completion, log10() returns the base 10 logarithm of \( x \).

If \( x \) is NaN, NaN is returned. If \( x \) is less than 0, -HUGE_VAL or NaN is returned, and errno is set to EDOM. If \( x \) is 0, -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 log10() function will fail if:

EDOM The value of \( x \) is negative.

The log10() function may fail if:

ERANGE The value of \( x \) is 0.

No other errors will occur.

USAGE
An application wishing to check for error situations should set errno to 0 before calling log10(). 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>
<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>

SEE ALSO
isnan(3M), log(3M), matherr(3M), pow(3M), attributes(5), standards(5)
The `log1p()` function computes \( \log_e(1.0 + x) \). The value of \( x \) must be greater than \(-1.0\).

Upon successful completion, `log1p()` returns the natural logarithm of \( 1.0 + x \).

If \( x \) is NaN, `log1p()` returns NaN.

If \( x \) is less than \(-1.0\), `log1p()` returns `-HUGE_VAL` or NaN and sets `errno` to `EDOM`.

If \( x \) is \(-1.0\), `log1p()` returns `-HUGE_VAL` and may set `errno` to `ERANGE`.

For exceptional cases, `matherr(3M)` tabulates the values to be returned as dictated by Standards other than XPG4.

The `log1p()` function will fail if:

- **EDOM** The value of \( x \) is less than \(-1.0\).

The `log1p()` function may fail and set `errno` to:

- **ERANGE** The value of \( x \) is \(-1.0\).

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>

See also `log(3M), matherr(3M), attributes(5), standards(5)`
NAME       log – natural logarithm function

SYNOPSIS   cc [ flag ... ] file ... -lm [ library ... ]
            #include <math.h>
            double log(double x);

DESCRIPTION The log() function computes the natural logarithm of \( x, \log_e(x) \). The value of \( x \) must be positive.

RETURN VALUES Upon successful completion, log() returns the natural logarithm of \( x \).

If \( x \) is NaN, NaN is returned.

If \( x \) is less than 0, -HUGE_VAL or NaN is returned and errno is set to EDOM.

If \( x \) is 0, -HUGE_VAL is returned and errno may be set to ERANGE.

In IEEE 754 mode (the -Xlibmieee cc 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, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS    The log() function will fail if:
            EDOM        The value of \( x \) is negative.

            The log() function may fail if:
            ERANGE      The value of \( x \) is 0.

            No other errors will occur.

USAGE      An application wishing to check for error situations should set errno to 0 before calling log(). 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>
<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>

SEE ALSO     exp(3M), isnan(3M), log10(3M), log1p(3M), matherr(3M), attributes(5), standards(5)
The `logb()` 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.

Upon successful completion, `logb()` returns the exponent of \( x \).

If \( x \) is 0.0, `logb()` returns `-HUGE_VAL` and sets `errno` to `EDOM`.

If \( x \) is ±Inf, `logb()` returns `+Inf`.

If \( x \) is NaN, `logb()` returns NaN.

For exceptional cases, `matherr(3M)` tabulates the values to be returned as dictated by various Standards.

The `logb()` function will fail if:

EDOM The \( x \) argument is 0.0.

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>

See also `ilogb(3M), matherr(3M), attributes(5)`
maillock(3MAIL)

NAME
maillock, mailunlock, touchlock – functions to manage lockfile(s) for user’s mailbox

SYNOPSIS
c [ flag ... ] file ... -lmail [ library ... ]
#include <maillock.h>

int maillock(const char *user, int retrycnt);

void mailunlock(void);

void touchlock(void);

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 retrycnt.

When the lockfile is no longer needed, it should be removed by calling
mailunlock().

The user 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
/var/mail/* user mailbox files
/var/mail/*/lock user mailbox lockfiles

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
libmail(3LIB), attributes(5)
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.
The System V Interface Definition, Third Edition (SVID3) specifies that certain `libm` functions call `matherr()` when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named `matherr()` in their programs. The `matherr()` function is of the form described above. When an exception occurs, a pointer to the exception structure `exc` will be passed to the user-supplied `matherr()` function. This structure, which is defined in the `<math.h>` header file, is as follows:

```c
struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
};
```

The `type` member is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):

- **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

Note that both `TLOSS` and `PLOSS` 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, `PLOSS` is never signaled. `TLOSS` is signaled for Bessel functions *only* to satisfy SVID3 requirements.

The `name` member points to a string containing the name of the function that incurred the exception. The `arg1` and `arg2` members are the arguments with which the function was invoked. `retval` is set to the default value that will be returned by the function unless the user’s `matherr()` sets it to a different value.

If the user’s `matherr()` function returns non-zero, no exception message will be printed, and `errno` will not be set.

When an application is built as a SVID3 conforming application (see `standards(5)`), if `matherr()` is not supplied by the user, the default `matherr` exception-handling mechanisms, summarized in the table below, will be invoked upon exception:
0.0 is usually returned, \texttt{errno} is set to \texttt{EDOM}, and a message is usually printed on standard error.

The largest finite single-precision number, \texttt{HUGE} of appropriate sign is returned, \texttt{errno} is set to \texttt{EDOM}, and a message is printed on standard error.

The largest finite single-precision number, \texttt{HUGE} of appropriate sign is usually returned, \texttt{errno} is set to \texttt{ERANGE}.

0.0 is returned, and \texttt{errno} is set to \texttt{ERANGE}.

0.0 is returned, \texttt{errno} is set to \texttt{ERANGE}, and a message is printed on standard error.

In general, \texttt{errno} is not a reliable error indicator in that it may be unexpectedly set by a function in a handler for an asynchronous signal.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\texttt{<math.h>} type & \texttt{DOMAIN} & \texttt{SING} & \texttt{OVERFLOW} & \texttt{UNDERFLOW} & \texttt{TLOSS} \\
\hline
\texttt{errno} & \texttt{EDOM} & \texttt{EDOM} & \texttt{ERANGE} & \texttt{ERANGE} & \texttt{ERANGE} \\
\hline
\texttt{IEEE Exception} & Invalid Operation & Division by Zero & Overflow & Underflow & \texttt{–} \\
\hline
\texttt{fp\_exception\_type} & \texttt{fp\_invalid} & \texttt{fp\_division} & \texttt{fp\_overflow} & \texttt{fp\_underflow} & \texttt{–} \\
\hline
\texttt{ACOS, ASIN} (&|x| > 1): & Md, 0.0 & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\hline
\texttt{ACOSH} (x < 1), ATANH (&|x| > 1): & NaN & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\hline
\texttt{ATAN2} (0,0): & Md, 0.0 & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\hline
\texttt{COSH, SINH}: & \texttt{–} & \texttt{–} & \texttt{±HUGE} & \texttt{–} & \texttt{–} \\
\hline
\texttt{EXP}: & \texttt{–} & \texttt{–} & \texttt{+HUGE} & 0.0 & \texttt{–} \\
\hline
\texttt{FMOD} (x,0): & x & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\hline
\texttt{HYPOT}: & \texttt{–} & \texttt{–} & \texttt{+HUGE} & \texttt{–} & \texttt{–} \\
\hline
\texttt{J0, J1, JN} (&|x| > \texttt{X\_TLOSS}): & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{Mt, 0.0} \\
\hline
\texttt{LGAMMA}: & \texttt{–} & \texttt{–} & \texttt{+HUGE} & \texttt{–} & \texttt{–} \\
\texttt{usual cases} & \texttt{–} & \texttt{–} & \texttt{+HUGE} & \texttt{–} & \texttt{–} \\
\texttt{(x = 0 or –integer)} & \texttt{–} & \texttt{Ms, +HUGE} & \texttt{–} & \texttt{–} & \texttt{–} \\
\texttt{LOG, LOG10}: & \texttt{Md, –HUGE} & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\texttt{(x < 0)} & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} & \texttt{–} \\
\hline
\end{tabular}
\end{table}
### 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 \( \infty \) 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 the `matherr()` may return results that differ from those in this table.

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 shall be returned. In particular, `pow(NaN, 0) = NaN`.

### Table

<table>
<thead>
<tr>
<th><code>&lt;math.h&gt;</code> type</th>
<th>DOMAIN</th>
<th>SING</th>
<th>OVERFLOW</th>
<th>UNDERFLOW</th>
<th>TLOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (x = 0) )</td>
<td>−</td>
<td>Ms, –HUGE</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>POW:</strong></td>
<td>−</td>
<td>−</td>
<td>±HUGE</td>
<td>±0.0</td>
<td>−</td>
</tr>
<tr>
<td>usual cases</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( (x &lt; 0)^{2} ) (y not an integer)</td>
<td>Md, 0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( 0^{2} 0 )</td>
<td>Md, 0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( 0^{2} (y &lt; 0) )</td>
<td>Md, 0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>REMAINDER</strong> (x,0): NaN</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>SCALB:</strong></td>
<td>−</td>
<td>−</td>
<td>±HUGE_VAL</td>
<td>±0.0</td>
<td>−</td>
</tr>
<tr>
<td><strong>SQRT</strong> (x &lt; 0): Md, 0.0</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>Y0, Y1, YN:</strong></td>
<td>Md, −HUGE</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( (x &lt; 0) )</td>
<td>Md, −HUGE</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( (x = 0) )</td>
<td>−</td>
<td>Md, −HUGE</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>( (x &gt; X_TLOSS) )</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>Mt, 0.0</td>
</tr>
</tbody>
</table>
<math.h> type | DOMAIN | SING | OVERFLOW | UNDERFLOW | TLOSS
---|---|---|---|---|---
errno | EDOM | EDOM | ERANGE | ERANGE | ERANGE
ACOS,\nASIN (|x| > 1): 0.0 | − | − | − | − | −
ATAN2 (0,0): 0.0 | − | − | − | − | −
COSH,\nSINH: | − | − | [+HUGE_VAL] | − | −
EXP: | − | − | [+HUGE_VAL] | [0.0] | −
FMOD (x,0): [NaN] | − | − | − | − | −
HYPOP: | − | − | [+HUGE_VAL] | − | −
J0, J1,\nJN (|x| > X_TLOSS): | − | − | − | − | [0.0]
LGAMMA: usual cases | − | − | [+HUGE_VAL] | − | −
(x = 0 or\n−integer) | − | +HUGE_VAL | − | − | −
LOG,\nLOG10: (x < 0) | −HUGE_VAL | − | − | − | −
(x = 0) | − | −HUGE_VAL | − | − | −
POW: usual cases | − | − | ±HUGE_VAL | ±0.0 | −
(x < 0) ** (y\nnot an\ninteger) 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 | − | − | −

Extended Library Functions 293
matherr(3M)

<table>
<thead>
<tr>
<th>&lt;math.h&gt; type</th>
<th>DOMAIN</th>
<th>SING</th>
<th>OVERFLOW</th>
<th>UNDERFLOW</th>
<th>TLOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &gt; X_TLOSS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Abbreviations**

[...] errno is not to be relied upon in all braced cases.

NaN IEEE NaN result and invalid operation exception.

HUGEINVAL IEEE result and division-by-zero exception.

X_TLOSS The value X_TLOSS is defined in <values.h>.

The ANSI/ISO-C standard covers a small subset of the CAE specification.

The following table summarizes the values returned in the exceptional cases.

<table>
<thead>
<tr>
<th>&lt;math.h&gt; type</th>
<th>DOMAIN</th>
<th>SING</th>
<th>OVERFLOW</th>
<th>UNDERFLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td>EDOM</td>
<td>EDOM</td>
<td>ERANGE</td>
<td>ERANGE</td>
</tr>
<tr>
<td>ACOS, ASIN (</td>
<td>x</td>
<td>&gt; 1)</td>
<td>0.0</td>
<td>–</td>
</tr>
<tr>
<td>ATAN2 (0,0)</td>
<td>0.0</td>
<td>–</td>
<td>–</td>
<td>+HUGE_VAL</td>
</tr>
<tr>
<td>EXP</td>
<td>–</td>
<td>–</td>
<td>+HUGE_VAL</td>
<td>0.0</td>
</tr>
<tr>
<td>FMOD (x,0)</td>
<td>NaN</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOG, LOG10:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x &lt; 0)</td>
<td>-HUGE_VAL</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>(x = 0)</td>
<td>–</td>
<td>-HUGE_VAL</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>POW:</td>
<td></td>
<td></td>
<td>±HUGE_VAL</td>
<td>±0.0</td>
</tr>
<tr>
<td>usual cases</td>
<td>–</td>
<td>–</td>
<td>±HUGE_VAL</td>
<td>±0.0</td>
</tr>
<tr>
<td>(x &lt; 0) ** (y</td>
<td>0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>not an integer)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 ** (y &lt; 0)</td>
<td>-HUGE_VAL</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SQRT (x &lt; 0)</td>
<td>0.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

**ABBREVIATIONS**

NaN IEEE NaN result and invalid operation exception.

HUGE_VAL IEEE result and division-by-zero exception.
EXAMPLE 1 Example of matherr() function
#include <stdio.h>
#include <stdlib.h>
#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:

<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>

SEE ALSO  attributes(5), standards(5)
m_create_layout(3LAYOUT)

NAME  m_create_layout – initialize a layout object

SYNOPSIS  cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

LayoutObject m_create_layout(const AttrObject attrobj, const char* modifier);

DESCRIPTION  The m_create_layout() function creates a LayoutObject associated with the
locale identified by attrobj.

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 attrobj. The memory for the LayoutObject is
allocated by m_create_layout(). The LayoutObject created has default layout
values. If the modifier argument is not NULL, the layout values specified by the modifier
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 attrobj 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
attrobj argument specifies a name that is usually associated with a locale category. If
attrobj is NULL, the created LayoutObject is associated with the current locale as set
by the setlocale(3C) function.

The modifier argument announces a set of layout values when the LayoutObject is
created.

RETURN VALUES  Upon successful completion, the m_create_layout() function returns a
LayoutObject for use in subsequent calls to m_*_layout() functions. Otherwise
the m_create_layout() function returns (LayoutObject) 0 and sets errno to
indicate the error.

ERRORS  The m_create_layout() function may fail if:

EBADF  The attribute object is invalid or the locale associated with the
attribute object is not available.

EINVAL  The modifier 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.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:
### m_create_layout(3LAYOUT)

<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>

**SEE ALSO** setlocale(3C), attributes(5)
md5(3EXT)

NAME
md5, md5_calc, MD5Init, MD5Update, MD5Final – MD5 digest functions

SYNOPSIS
ce [ flag ... ] file ... -lmd5 [ library ... ]
#include <md5.h>

#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);

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 md5_calc() function computes an MD5 digest on a single message block. The
ilen-byte block is pointed to by input, and the 16-byte MD5 digest is written to output.

MD5Init(), MD5Update(), MD5Final() The MD5Init(), MD5Update(), and MD5Final() 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 MD5Init(), one or more calls to
MD5Update(), and one call to MD5Final(), in that order.

The MD5Init() function initializes the MD5 context structure pointed to by context.

The MD5Update() function computes a partial MD5 digest on the ilen-byte message
block pointed to by input, and updates the MD5 context structure pointed to by context
accordingly.

The MD5Final() function generates the final MD5 digest, using the MD5 context
structure pointed to by context; the 16-byte MD5 digest is written to output. After
calling MD5Final(), the state of the context structure is undefined; it must be
reinitialized with MD5Init() 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.

```c
int AuthenticateMsg(unsigned char *auth_buffer, struct iovec *messageIov,
                      unsigned int num_buffers)
{...
```
EXAMPLE 1 Authenticate a message found in multiple buffers

```c
MD5_CTX md5_context;
unsigned int i;

MD5Init(&md5_context);
for(i=0; i<num_buffers; i++)
{
    MD5Update(&md5_context, messageIov->iov_base,
              messageIov->iov_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.

```c
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:

<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>

SEE ALSO libmd5(3LIB)

m_destroy_layout(3LAYOUT)

NAME | m_destroy_layout – destroy a layout object
SYNOPSIS | cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

int m_destroy_layout(const LayoutObject layoutobject);

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:
EBADF | The attribute object is erroneous.
EFAULT | Errors occurred while processing the request.

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>

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**
```
cc [ flag ...] file ... -lvolmgt [ library ...]
#include <volmgt.h>

char *media_findname(char *start);
```

**DESCRIPTION**
media_findname() converts the supplied start string into an absolute pathname that can then be used to access a particular piece of media.

The start parameter can be one of the following types of specifications:

- **/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.

- **volmgt_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.
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).

```c
(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.

```c
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Unsafe</td>
</tr>
</tbody>
</table>

**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)`).
media_getattr(3VOLMGT)

NAME
media_getattr, media_setattr – get and set media attributes

SYNOPSIS
c [ 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);

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>
<thead>
<tr>
<th>Attribute</th>
<th>Writable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-access</td>
<td>RO</td>
<td>&quot;seq&quot;, &quot;rand&quot;</td>
<td>sequential or random access</td>
</tr>
<tr>
<td>s-density</td>
<td>RO</td>
<td>&quot;low&quot;, &quot;medium&quot;, &quot;high&quot;</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>&quot;true&quot;, &quot;false&quot;</td>
<td>whether or not media is manually ejectable</td>
</tr>
<tr>
<td>s-rmoneject</td>
<td>R/W</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
<td>should media access points be removed from database upon ejection</td>
</tr>
<tr>
<td>s-enxio</td>
<td>R/W</td>
<td>&quot;true&quot;, &quot;false&quot;</td>
<td>if set return ENXIO when media access attempted</td>
</tr>
</tbody>
</table>

Properties can also be defined by the user. In this case the value can be any string the user wishes.

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.
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, `vold`, is not running
- **EINTR** The routine was interrupted by the user before finishing

### 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:

```c
if (media_getattr("/vol/rdsk/fred", "s-mejectable") != NULL) {
    (void) printf(""fred"") must be manually ejected\n\n); } else {
    (void) printf("software can eject "fred\n\n"); }
```

This example shows setting the `s-enxio` property for the floppy volume currently in the first floppy drive:

```c
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\n"); }
```

### 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>

### 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)`).
media_getid(3VOLMGT)

NAME

media_getid – return the id of a piece of media

SYNOPSIS

c[flag ...] file ... -lvolgmt [library ...]

#include <volmgmt.h>
ulonglong_t media_getid(char *vol_path);

DESCRIPTION

media_getid() returns the id of a piece of media. Volume Management must be
running. See volmgt_running(3VOLMGT).

PARAMETERS

vol_path Path to the block or character special device.

RETURN VALUES

media_getid() returns the id of the volume. This value is unique for each volume. If
media_getid() returns 0, the path 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 path, and then returns the id for the piece of
media.

char *path;
...

if (volmgt_running()) {
    if (volmgt_ownspath(path)) {
        (void) printf("id of %s is %lld\n", path, media_getid(path));
    }
}

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 path in its name space, or Volume
Management is not running.

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>

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
cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

int m_getvalues_layout(const LayoutObject layout_object, LayoutValues values, int *index_returned);

DESCRIPTION
The m_getvalues_layout() function queries the current setting of layout values within a LayoutObject.

The layout_object argument specifies a LayoutObject returned by the m_create_layout(3LAYOUT) function.

The values argument specifies the list of layout values that are to be queried. Each value element of a LayoutValueRec must point to a location where the layout value is stored. That is, if the layout value is of type T, the argument must be of type T*. The values are queried from the LayoutObject 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 QueryValueSize 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 m_getvalues_layout() function returns 0. If any value cannot be queried, the index of the value causing the error is returned in index_returned, −1 is returned and errno is set to indicate the error.

ERRORS
The m_getvalues_layout() function may fail if:

EINVAL
The layout value specified by index_returned is unknown, its value is invalid, or the layout_object argument is invalid. In the case of an invalid layout_object argument, the value returned in index_returned is −1.

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>

SEE ALSO
m_create_layout(3LAYOUT), attributes(5)
mkdirp(3GEN)

NAME
mkdirp, rmdirp – create or remove directories in a path

SYNOPSIS
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int mkdirp(const char *path, mode_t mode);
int rmdirp(char *dir, char *dir1);

DESCRIPTION
The mkdirp() function creates all the missing directories in path with mode. See chmod(2) for the values of mode.

The rmdirp() function removes directories in path dir. 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 dir1.

RETURN VALUES
If path already exists or if a needed directory cannot be created, mkdirp() returns -1 and sets errno to one of the error values listed for mkdir(2). It returns zero if all the directories are created.

The rmdirp() 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.

```c
/* create scratch directories */
if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == -1) {
    fprintf(stderr, "cannot create directory");
    exit(1);
}
chdir("/tmp/sub1/sub2/sub3");
.
.
/* cleanup */
chdir("/tmp");
rmdirp("sub1/sub2/sub3");
```

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>

SEE ALSO
chmod(2), mkdir(2), rmdir(2), malloc(3C), attributes(5)

NOTES
mkdirp() uses malloc(3C) to allocate temporary space for the string.
When compiling multithreaded applications, the \texttt{REENTRANT} flag must be defined on the compile line. This flag should only be used in multithreaded applications.
mp(3MP)

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
c make [ 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);

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 mp_itom(n), which sets the initial value to n. Alternatively, mp_xtom(a) may be used to initialize a MINT from a string of hexadecimal digits. mp_mfree(a) may be used to release the storage allocated by the mp_itom(a) and mp_xtom(a) routines.

The mp_madd(a,b,c), mp_msub(a,b,c) and mp_mult(a,b,c) functions assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. The mp_mdiv(a,b,q,r) function assigns the quotient and remainder, respectively, to its third and fourth arguments. The mp_sdiv(a,n,q,r) function is similar to mp_mdiv(a,b,q,r) except that the divisor is an ordinary integer. The mp_msqrt(a,b,r) function produces the square root and remainder of its first argument. The mp_mcmp(a,b) 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 mp_rpow(a,n,b) function raises a to the nth power and assigns this value to b. The
mp_pow(a, b, c, d) function raises a to the bth 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:

<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>

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).
m_setvalues_layout(3LAYOUT)

NAME  m_setvalues_layout – set layout values of a LayoutObject

SYNOPSIS  cc [ flag... ] file... -llayout [ library... ]
           #include <sys/layout.h>
           int m_setvalues_layout(LayoutObject layout_object, const LayoutValues values, int *index_returned);

DESCRIPTION  The m_setvalues_layout() function changes the layout values of a LayoutObject.

            The layout_object argument specifies a LayoutObject returned by the
            m_create_layout(3LAYOUT) function.

            The values 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 m_setvalues_layout() 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 errno 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 index_returned.

ERRORS  The m_setvalues_layout() function may fail if:

            EINVAL  The layout value specified by index_returned is unknown, its value
                    is invalid, or the layout_object 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 m_setvalues_layout() function.

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>

SEE ALSO  m_create_layout(3LAYOUT), attributes(5)
m_transform_layout(3LAYOUT)

NAME
m_transform_layout – layout transformation

SYNOPSIS
cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

int m_transform_layout(LayoutObject layout_object, const char *InpBuf,
const size_t ImpSize, const void *OutBuf, size_t *Outsize, size_t *InpToOut,
size_t *OutToInp, unsigned char *Property, size_t *InpBufIndex);

DESCRIPTION
The m_transform_layout() 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
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 m_transform_layout() function transforms the input text in InpBuf according
to the current layout values in layout_object. Any layout value whose value type is
LayoutTextDescriptor describes the attributes of the InpBuf and OutBuf
arguments. If the attributes are the same for both InpBuf and OutBuf, a null
transformation is performed with respect to that specific layout value.

The InpBuf argument specifies the source text to be processed. The InpBuf may not be
NULL, unless there is a need to reset the internal state.

The ImpSize argument is the number of bytes within InpBuf to be processed by the
transformation. Its value will not change after return from the transformation. ImpSize
set to −1 indicates that the text in InpBuf is delimited by a null code element. If ImpSize
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:

OutBuf Any transformed data is stored in OutBuf, converted to
ShapeCharset.

Outsize The number of bytes in OutBuf.

ImpToOut A cross-reference from each InpBuf code element to the
transformed data. The cross-reference relates to the data in InpBuf
starting with the first element that InpBufIndex points to (and not
necessarily starting from the beginning of the InpBuf).

OutToInp A cross-reference to each InpBuf code element from the
transformed data. The cross-reference relates to the data in InpBuf
starting with the first element that InpBufIndex points to (and not
necessarily starting from the beginning of the InpBuf).

Extended Library Functions 313
m_transform_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 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 NestingLevels 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.
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 nth value in `InpToOut` corresponds to the nth byte in `InpBuf`. This value is the index (in units of bytes) in `OutBuf` that identifies the transformed `ShapeCharset` element of the nth 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 nth value in `OutToInp` corresponds to the nth byte in `OutBuf`. This value is the index (in units of bytes) of `OutBuf` that identifies the logical code element of the nth 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.
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`.

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 `InpBufIndex` 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
  `ShapeCharset` layout value, or is an invalid code element not
  defined by the codeset of the locale of `layout_object`. The
  `mbtowc()` and `wctomb()` functions, when used in the same locale as the
  `LayoutObject`, 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 `OutSize` contains `NULL`.
- **ERANGE** More than 15 embedding levels are in source text or `InpBuf` contains
  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 `InpBufIndex`.

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.

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`.

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>
m_transform_layout(3LAYOUT)

SEE ALSO  attributes(5)
m_wtransform_layout(3LAYOUT)

NAME  m_wtransform_layout – layout transformation for wide character strings

SYNOPSIS

cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

int m_wtransform_layout(LayoutObject layout_object, const wchar_t *InpBuf, const size_t ImpSize, const void *OutBuf, size_t *Outsize, size_t *InpToOut, size_t *OutToInp, unsigned char *Property, size_t *InpBufIndex);

DESCRIPTION

The m_wtransform_layout() 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 m_wtransform_layout() function transforms the input text in InpBuf according to the current layout values in layout_object. Any layout value whose value type is LayoutTextDescriptor describes the attributes of the InpBuf and OutBuf arguments. If the attributes are the same for both InpBuf and OutBuf, a null transformation is performed with respect to that specific layout value.

The InpBuf argument specifies the source text to be processed. The InpBuf may not be NULL, unless there is a need to reset the internal state.

The ImpSize argument is the number of bytes within InpBuf to be processed by the transformation. Its value will not change after return from the transformation. ImpSize set to −1 indicates that the text in InpBuf is delimited by a null code element. If ImpSize 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:

OutBuf Any transformed data is stored in OutBuf, converted to ShapeCharset.

Outsize The number of wide characters in OutBuf.

InpToOut A cross-reference from each InpBuf code element to the transformed data. The cross-reference relates to the data in InpBuf starting with the first element that InpBufIndex points to (and not necessarily starting from the beginning of the InpBuf).

OutToInp A cross-reference to each InpBuf code element from the transformed data. The cross-reference relates to the data in InpBuf starting with the first element that InpBufIndex points to (and not necessarily starting from the beginning of the InpBuf).
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.
The `m_wtransform_layout()` function may fail if:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2BIG</td>
<td>The output buffer is full and the source text is not entirely processed.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The layout values are set to a meaningless combination or the layout object is not valid.</td>
</tr>
<tr>
<td>EILSEQ</td>
<td>Transformation stopped due to an input code element that cannot be shaped or is invalid. The <code>InpBufIndex</code> 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 <code>ShapeCharset</code> layout value, or is an invalid code element not defined by the codeset of the locale of <code>layout_object</code>. The <code>mbtowc()</code> and <code>wctomb()</code> functions, when used in the same locale as the <code>LayoutObject</code>, can be used to determine if the code element is valid.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or <code>OutSize</code> contains NULL.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>More than 15 embedding levels are in source text or <code>InpBuf</code> 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 <code>InpBufIndex</code>. 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>
</tbody>
</table>

**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_wtransform_layout()` might be unpredictable or implementation-specific with `errno` set to `EBADF`.

**Examples**

**Example 1** Shaping and reordering input string into output buffer

The following example illustrated what the different arguments of `m_wtransform_layout()` look like when a string in `InpBuf` is shaped and reordered into `OutBuf`. Upper-case letters in the example represent left-to-right letters while lower-case letters represent right-to-left letters. `xyz` represents the shapes of `cde`.

<table>
<thead>
<tr>
<th>Position</th>
<th>InpBuf:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123456789</td>
<td>AB cde 12z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position</th>
<th>OutBuf:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123456789</td>
<td>AB 12 zyxZ</td>
</tr>
</tbody>
</table>

| Position | 0123456789 |
EXAMPLE 1 Shaping and reordering input string into output buffer

OutToInp: 0127865439
Position: 0123456789
Property.NestLevel: 0001111220
Property.CellBdry: 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)

<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>

SEE ALSO attributes(5)
newDmiOctetString(3DMI)

NAME newDmiOctetString – create DmiOctetString in dynamic memory

SYNOPSIS cc [ flag ... ] file ... -ldmi -lnsl -lrwtool [ library ... ]
   #include <dmi/util.hh>
   DmiOctetString_t *newDmiOctetString(DmiOctetString_t *str);

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>
<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>

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

NAME
newDmiString – create DmiString in dynamic memory

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

DmiString_t *newDmiString(char *str);

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:

<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>

SEE ALSO
freeDmiString(3DMI), libdmi(3LIB), attributes(5)
nextafter() returns the next representable double-precision floating-point value following \( x \) in the direction of \( y \).

If \( x \) or \( y \) is NaN, then \( \text{nextafter}() \) returns NaN.

If \( x \) is finite and the correct function value would overflow, \( \text{nextafter}() \) returns \( \pm \text{HUGE\_VAL} \) (according to the sign of \( x \)) and sets \( \text{errno} \) to \( \text{ERANGE} \).

The \( \text{nextafter}() \) function will fail if:

- \( \text{ERANGE} \): The correct value would overflow.

See also 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>

See also attributes(5),
nlist(3ELF)

NAME  nlist – get entries from name list

SYNOPSIS  cc [ flag... ] file ... -lelf [ library ... ]
#include <nlist.h>

int nlist(const char *filename, struct nlist *nl);

DESCRIPTION  nlist() examines the name list in the executable file whose name is pointed to by filename, and selectively extracts a list of values and puts them in the array of nlist() structures pointed to by nl. The name list nl 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 -g 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 n_name are set to 0.

This function is useful for examining the system name list kept in the file /dev/ksyms. 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>
<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>

SEE ALSO  cc(1B), elf(3ELF), kvm_nlist(3KVM), kvm_open(3KVM), libelf(3LIB), a.out(4), attributes(5), ksym(7D), mem(7D)
NOTE(3EXT)

NAME
NOTE, _NOTE -- annotate source code with info for tools

SYNOPSIS
#include <note.h>

NOTE(NoteInfo);

or
#include<sys/note.h>

_NOTE(NoteInfo);

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 lock_lint).

NOTE(MUTEX_PROTECTS_DATA(foo_lock, foo_list Foo))

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, NOTE(x) 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:

x = y + NOTE(...) z;
typedef NOTE(...) unsigned int uint;

While NOTE and _NOTE 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 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:

<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>

SEE ALSO

note(4), attributes(5)
nvlist_add_boolean(3NVPAIR)

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
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);

PARAMETERS
nvl The nvlist_t (name-value pair list) to be processed.
name Name of the nvpair (name-value pair).
nelem Number of elements in value (that is, array size).

330 man pages section 3: Extended Library Functions • Last Revised 10 Apr 2001
These functions add a new name-value pair to an nvlist_t. The uniqueness of
nvpair name and data types follows the nvflag argument specified for
nenvlist_alloc(). See nvlist_alloc(3NVPAIR).

If NV_UNIQUE_NAME was specified for nvflag, existing nvpairs with matching names
are removed before the new nvpair is added.

If NV_UNIQUE_NAME_TYPE was specified for nvflag, existing nvpairs with matching
names and data types are removed before the new nvpair is added.

If neither was specified for nvflag, the new nvpair is unconditionally added at the
end of the list. The library preserves the order of the name-value pairs across packing,
unpacking, and duplication.

These functions return 0 on success and an error value on failure.

These functions will fail if:

EINVAL There is an invalid argument.
ENOMEM There is insufficient memory.

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>

libnvpair(3NVPAIR), attributes(5)
nvlist_alloc(3NVPAIR)

NAME        nvlist_alloc, nvlist_free, nvlist_size, nvlist_pack, nvlist_unpack, nvlist_dup -- manage a name-value pair list

SYNOPSIS    

cc [flag ...] file ... -lnvpair [library ...]
#include <libnvpair.h>

int nvlist_alloc(nvlist_t **nvlp, 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 **nvlp, int flag);
int nvlist_dup(nvlist_t *nvl, nvlist_t **nvlp, int flag);

PARAMETERS   

nvlp        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 nvlp 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:

    NV_ENCODE_NATIVE        Straight bcopy() as described in bcopy(3C).
    NV_ENCODE_XDR           Use XDR encoding, suitable for sending to another host.
The `nvlist_pack()` function packs `nvl` 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 `nvl` and updates `nvlp` to point to the copy.

**RETURN VALUES**

These functions return 0 on success and an error value on failure.

**ERRORS**

All five 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.

```c
/*
 * 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 */
    /* 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", strs[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", strs, 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));
    nvlist_free(nvl);
    return (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);
    }

    /* SQL */
    rv = read(fd, buf, buflen);
    if (rv < buflen) {
        fd = close(fd);
        return (-1);
    }

    /* pack into contig. memory */
    err = nvlist_pack(nvl, buf, buflen, encode, 0);
    if (err) {
        (void) printf("nvlist_pack: %s\n", strerror(err));
        return (err);
    }

    /* free the original list */
    nvlist_free(nvl);
    return (err);
}

/* SQL */

 nvlist_alloc(3NVPAIR)
(void) fstat(fd, &sbuf);
*buflen = sbuf.st_size;
*buf = malloc(*buflen);
if (*buf == NULL) {
    (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]);
            err = 1;
            break;
        }

    if (get_nvlist_buf(file, &buf, &buflen) != 0) {
        (void) printf("cannot get packed nvlist buffer\n");
        return;
    }
}
nvlist_alloc(3NVPAIR)

/* unpack into an nvlist_t */
err = nvlist_unpack(buf, buflen, &nvl, 0);
if (err) {
    (void) printf("nvlist_unpack(): %s\n",
    strerror(err));
    return;
}

/* selectively print out attributes */
nvlist_lookup_and_print(nvl);
return;

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>

SEE ALSO
libnvpair(3NVPAIR), attributes(5)
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

#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_int32(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);

PARAMETERS

nvl The nvlist_t to be processed.
name Name of the name-value pair to search.
nelem Address to store the number of elements in value.
nvlist_lookup_boolean(3NVPAIR)

val Address to store the starting address of the value.

DESCRIPTION These functions find the nvpair (name-value pair) that matches the name and type as indicated by the interface name. If one is found, nelem and val are modified to contain the number of elements in value and the starting address of data, respectively.

These functions work for nvlists (lists of name-value pairs) allocated with NV_UNIQUE_NAME or NV_UNIQUE_NAME_TYPE specified in nvlist_alloc(). (See nv_list_alloc(3NVPAIR).) If this is not the case, the function returns ENOTSUP because the list potentially contains multiple nvpairs 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 nvlist_free() is called on nvl.

RETURN VALUES Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS These functions will fail if:

EINVAL There is an invalid argument.
ENOENT No matching name-value pair is found
ENOTSUP An encode/decode method 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>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO libnvpair(3NVPAIR), attributes(5)
nvlist_next_nvpair(3NVPAIR)

NAME
nvlist_next_nvpair, nvpair_name, nvpair_type – return data regarding name-value pairs

SYNOPSIS
cce [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);

PARAMETERS
nvl The nvlist_t to be processed.
nvpair 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:

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

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:

<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>

SEE ALSO

libnvpair(3NVPAIR), attributes(5)
nvlist_remove(3NVPAIR)

NAME
nvlist_remove, nvlist_remove_all — remove name-value pairs

SYNOPSIS
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);

PARAMETERS
nvl The nvlist_t to be processed.
name Name of the name-value pair to be removed.
type 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>

SEE ALSO
libnvpair(3NVPAIR), attributes(5)
nvpair_value_byte(3NVPAIR)

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
c [flag ...] file ... -lvpair [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);

PARAMETERS
	nvpair Name-value pair to be processed.
nelem Address to store the number of elements in value.
val Address to store the value or the starting address of the array value.
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)`. 

These functions return 0 on success and an error value on failure.

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:

<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>

**SEE ALSO**

`libnvpair(3LIB)`, `attributes(5)`
p2open(3GEN)

NAME
p2open, p2close - open, close pipes to and from a command

SYNOPSIS
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>
int p2open(const char *cmd, FILE *fp[2]);
int p2close(FILE *fp[2]);

DESCRIPTION
p2open() forks and execs a shell running the command line pointed to by cmd. On
return, fp[0] points to a FILE pointer to write the command’s standard input and
fp[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.
#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); }

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

```c
#include <security/pam_appl.h>
cc [ flag... ] file ... -lpam [ library ... ]
```

**DESCRIPTION**

The PAM framework, libpam, 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 `pam_`.

The first category contains functions for establishing and terminating an authentication activity, which are `pam_start(3PAM)` and `pam_end(3PAM)`. The functions `pam_set_data(3PAM)` and `pam_get_data(3PAM)` maintain module specific data. The functions `pam_set_item(3PAM)` and `pam_get_item(3PAM)` maintain state information. `pam_strerror(3PAM)` 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, `pam_authenticate(3PAM)` and `pam_setcred(3PAM)`.

The third category of PAM interfaces is account management. The function `pam_acct_mgmt(3PAM)` 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 `pam_open_session(3PAM)` and `pam_close_session(3PAM)`.

The fifth category consists of the function that changes authentication tokens, `pam_chauthtok(3PAM)`. 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 `pam_putenv(3PAM)`, `pam_getenv(3PAM)`, and `pam_getenvlist(3PAM)`.

The `pam_*( )` interfaces are implemented through the library libpam. 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 `pam_sm_` prefix. The only difference between the `pam_sm_*( )` interfaces and their
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 `pam_start()`. `pam_start()` 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 `pam_authenticate()` to authenticate a particular user, and `pam_acct_mgmt()` 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 `pam_setcred()` to set any user credentials associated with the authentication service. Within one authentication transaction (between `pam_start()` and `pam_end()`), all calls to the PAM interface should be made with the same authentication handle returned by `pam_start()`. 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 `pam_authenticate()`, service modules may store data in the handle that is intended for use by `pam_setcred()`.

To perform session management, applications call `pam_open_session()`. Specifically, the system may want to store the total time for the session. The function `pam_close_session()` closes the current session.

When necessary, applications can call `pam_get_item()` and `pam_set_item()` to access and to update specific authentication information. Such information may include the current username.

To terminate an authentication transaction, the application simply calls `pam_end()`, which frees previously allocated space used to store authentication information.

Application–Authentication Interface

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, `conv()`, along with any associated application data pointers, through a `pam_conv` structure to the authentication service when it initiates an authentication transaction, via a call to `pam_start()`. The service 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.

Stacking Multiple Schemes

The PAM architecture enables authentication by multiple authentication services through `stacking`. System entry applications, such as `login(1)`, 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, `pam.conf(4)`. A system administrator determines this ordering, and also determines whether the same password can be used for all authentication services.
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.

The PAM functions may return one of the following generic values, or one of the values defined in the specific man pages:

- **PAM_SUCCESS**: The function returned successfully.
- **PAM_OPEN_ERR**: `dlopen()` failed when dynamically loading a service module.
- **PAM_SYMBOL_ERR**: Symbol not found.
- **PAM_SERVICE_ERR**: Error in service module.
- **PAM_SYSTEM_ERR**: System error.
- **PAM_BUF_ERR**: Memory buffer error.
- **PAM_CONV_ERR**: Conversation failure.
- **PAM_PERM_DENIED**: Permission denied.

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>MT Level</td>
<td>MT-Safe with exceptions</td>
</tr>
</tbody>
</table>

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
```c
#include <security/pam_appl.h>

int pam_acct_mgmt(pam_handle_t *pamh, int flags);
```

DESCRIPTION
The `pam_acct_mgmt()` 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 `pam_authenticate(3PAM)`.

The `pamh` argument is an authentication handle obtained by a prior call to `pam_start()`. The following flags may be set in the `flags` field:

- **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.

Upon successful completion, `PAM_SUCCESS` is returned. In addition to the error return values described in `pam(3PAM)`, the following values may be returned:

- **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 `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>

SEE ALSO
`pam(3PAM), pam_authenticate(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.
pam_authenticate(3PAM)

NAME pam_authenticate – perform authentication within the PAM framework

SYNOPSIS

cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_authenticate(pam_handle_t *pamh, int flags);

DESCRIPTION

The pam_authenticate() 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 pam_start() or pam_set_item().

The following flags may be set in the flags field:

- **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:

- **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 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>

SEE ALSO

pam(3PAM), pam_open_session(3PAM), pam_set_item(3PAM), pam_setcred(3PAM), pam_start(3PAM), libpam(3LIB), attributes(5)
In the case of authentication failures due to an incorrect username or password, it is the responsibility of the application to retry `pam_authenticate()` and to maintain the retry count. An authentication service module may implement an internal retry count and return an error `PAM_MAXTRIES` if the module does not want the application to retry.

If the PAM framework cannot load the authentication module, then it will return `PAM_ABORT`. 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, `pam_authenticate()` requires that the user re-authenticate each of the services.

A null authentication token in the authentication database will result in successful authentication unless `PAM_DISALLOW_NULL_AUTHTOK` was specified. In such cases, there will be no prompt to the user to enter an authentication token.

The interfaces in `libpam` 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
ec [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_chauthtok(pam_handle_t *pamh, const int flags);

DESCRIPTION
The pam_chauthtok() function is called to change the authentication token associated with a particular user referenced by the authentication handle pamh.

The following flag may be passed in to pam_chauthtok():

PAM_SILENT The password service should not generate any messages.
PAM_CHANGE_EXPIRED_AUTHTOK The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.

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 pam.conf(4).

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:

PAM_PERM_DENIED No permission.
PAM_AUTHTOK_ERR Authentication token manipulation error.
PAM_AUTHTOK_RECOVERY_ERR Authentication information cannot be recovered.
PAM_AUTHTOK_LOCK_BUSY Authentication token lock busy.
PAM_AUTHTOK_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 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>

SEE ALSO
login(1), passwd(1), pam(3PAM), pam_authenticate(3PAM), pam_start(3PAM), attributes
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()` function 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

pam_getenv – returns the value for a PAM environment name

SYNOPSIS

```c
#include <security/pam_appl.h>

char *pam_getenv(pam_handle_t *pamh, const char *name);
```

DESCRIPTION

The `pam_getenv()` function searches the PAM handle `pamh` for a value associated with `name`. If a value is present, `pam_getenv()` makes a copy of the value and returns a pointer to the copy back to the calling application. If no such entry exists, `pam_getenv()` returns `NULL`. It is the responsibility of the calling application to free the memory returned by `pam_getenv()`.

RETURN VALUES

If successful, `pam_getenv()` returns a copy of the value associated with `name` in the PAM handle; otherwise, it returns a `NULL` pointer.

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 with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO

`pam(3PAM), pam_getenvlist(3PAM), pam_putenv(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.
pam_getenvlist(3PAM)

NAME
pam_getenvlist – returns a list of all the PAM environment variables

SYNOPSIS
cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

char ***pam_getenvlist (pam_handle_t *pamh);

DESCRIPTION
The pam_getenvlist() function returns a list of all the PAM environment variables stored in the PAM handle pamh. The list is returned as a null-terminated array of pointers to strings. Each string contains a single PAM environment variable of the form name=value. The list returned is a duplicate copy of all the environment variables stored in pamh. It is the responsibility of the calling application to free the memory returned by pam_getenvlist().

RETURN VALUES
If successful, pam_getenvlist() returns in a null-terminated array a copy of all the PAM environment variables stored in pamh. Otherwise, pam_getenvlist() returns a null pointer.

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 with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO
pam(3PAM), pam_getenv(3PAM), pam_putenv(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.
pam_get_user(3PAM)

NAME
pam_get_user – PAM routine to retrieve user name

SYNOPSIS
cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>

int pam_get_user(pam_handle_t *pamh, char **user, const char *
    *prompt);

DESCRIPTION
The pam_get_user() 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
pam_start() or pam_set_item(), the PAM conversation function will be used to
prompt the user for the user name with the string "prompt". If prompt is NULL, then
pam_get_item() 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:

Please enter user name:

After the user name is gathered by the conversation function, pam_set_item() is
called to set the value of PAM_USER. 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 pamAuthenticate(). 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. Applications that call
pamAuthenticate() multiple times should set the value of PAM_USER to NULL
with pam_set_item() before calling pamAuthenticate(), if they want the user
to be prompted for a new user name each time. The value of user retrieved by
pam_get_user() should not be modified or freed. The item will be released by
pam_end().

RETURN VALUES
Upon success, pam_get_user() 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:

<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>

SEE ALSO
pam(3PAM), pamAuthenticate(3PAM), pam_end(3PAM), pam_get_item(3PAM),
pam_set_item(3PAM), pam_sm(3PAM), pam_smAuthenticate(3PAM),
pam_start(3PAM), attributes(5)
The interfaces in *libpam* 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 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);

DESCRIPTION The pam_open_session() function is called after a user has been successfully authenticated. See pam_authenticate(3PAM) and pam_acct_mgmt(3PAM). It is used to notify the session modules that a new session has been initiated. All programs that use the pam(3PAM) library should invoke pam_open_session() when beginning a new session. Upon termination of this activity, pam_close_session() should be invoked to inform pam(3PAM) that the session has terminated.

The pamh argument is an authentication handle obtained by a prior call to pam_start(). The following flag may be set in the flags field for pam_open_session() and pam_close_session():

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 pam(3PAM), the following value may be returned on error:

PAM_SESSION_ERR Cannot make or remove an entry for the specified session.

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>

SEE ALSO getutxent(3C), pam(3PAM), pam_acct_mgmt(3PAM), pam_authenticate(3PAM), pam_start(3PAM), attributes(5)

NOTES In many instances, the pam_open_session() and pam_close_session() 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 pam_close_session() with an earlier call to pam_open_session(). 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, pamh. The call to pam_open_session() should precede UTMP/WTMP entry management, and the call to pam_close_session() should follow UTMP/WTMP exit management.
The interfaces in libpam are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.
pam_putenv(3PAM)

NAME     pam_putenv – change or add a value to the PAM environment

SYNOPSIS  cc [ flag ... ] file ... -lpam [ library ... ]
          #include <security/pam_appl.h>
          int pam_putenv(pam_handle_t *pamh, const char *name_value);

DESCRIPTION The pam_putenv() function sets the value of the PAM environment variable name equal to value either by altering an existing PAM variable or by creating a new one.

The name_value argument points to a string of the form name=value. A call to pam_putenv() does not immediately change the environment. All name_value pairs are stored in the PAM handle pamh. An application such as login(1) may make a call to pam_getenv(3PAM) or pam_getenvlist(3PAM) to retrieve the PAM environment variables saved in the PAM handle and set them in the environment if appropriate. login will not set PAM environment values which overwrite the values for SHELL, HOME, LOGNAME, MAIL, CDPATH, IFS, and PATH. Nor will login set PAM environment values which overwrite any value that begins with LD_.

If name_value equals NAME=, then the value associated with NAME in the PAM handle will be set to an empty value. If name_value equals NAME, then the environment variable NAME will be removed from the PAM handle.

RETURN VALUES The pam_putenv() function may return one of the following values:

- PAM_SUCCESS The function returned successfully.
- PAM_OPEN_ERR dlopen() failed when dynamically loading a service module.
- PAM_SYMBOL_ERR Symbol not found.
- PAM_SERVICE_ERR Error in service module.
- PAM_SYSTEM_ERR System error.
- PAM_BUF_ERR Memory buffer error.
- PAM_CONV_ERR Conversation failure.
- PAM_PERM_DENIED Permission denied.

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 with exceptions</td>
</tr>
</tbody>
</table>

SEE ALSO dlopen(3DL), pam(3PAM), pam_getenv(3PAM), pam_getenvlist(3PAM), libpam(3LIB), attributes(5)
The interfaces in libpam are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.
The `pam_setcred()` 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 `pam_authenticate(3PAM)`, `pam_acct_mgmt(3PAM)`, and `pam_open_session(3PAM)`.

The user is specified by a prior call to `pam_start()` or `pam_set_item()`, and is referenced by the authentication handle, `pamh`. The following flags may be set in the `flags` field. Note that the first four flags are mutually exclusive:

- **PAM_ESTABLISH_CRED**: Set user credentials for an authentication service.
- **PAM_DELETE_CRED**: Delete user credentials associated with an authentication service.
- **PAM_REINITIALIZE_CRED**: Reinitialize user credentials.
- **PAM_REFRESH_CRED**: Extend lifetime of user credentials.
- **PAM_SILENT**: Authentication service should not generate any messages.

If no flag is set, **PAM_ESTABLISH_CRED** is used as the default.

**RETURN VALUES**
Upon success, `pam_setcred()` returns **PAM_SUCCESS**. In addition to the error return values described in `pam(3PAM)` the following values may be returned upon error:

- **PAM_CRED_UNAVAIL**: Underlying authentication service can not retrieve user credentials unavailable.
- **PAM_CRED_EXPIRED**: User credentials expired.
- **PAM_USER_UNKNOWN**: User unknown to underlying authentication service.
- **PAM_CRED_ERR**: Failure setting user credentials.

**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>
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
#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);

DESCRIPTION
The pam_set_data() and pam_get_data() functions allow PAM service modules
to access and update module specific information as needed. These functions should
not be used by applications.

The pam_set_data() function stores module specific data within the PAM handle
pamh. The module_data_name argument uniquely identifies the data, and the data
argument represents the actual data. The module_data_name argument should be
unique across all services.

The cleanup function frees up any memory used by the data after it is no longer
needed, and is invoked by pam_end(). The cleanup function takes as its arguments a
pointer to the PAM handle, pamh, a pointer to the actual data, data, and a status code,
pam_end_status. The status code determines exactly what state information needs to be
purged.

If pam_set_data() is called and module data already exists from a prior call to
pam_set_data() under the same module_data_name, then the existing data is replaced
by the new data, and the existing cleanup function is replaced by the new cleanup
function.

The pam_get_data() function retrieves module-specific data stored in the PAM
handle, pamh, identified by the unique name, module_data_name. The data argument is
assigned the address of the requested data. The data retrieved by pam_get_data() should not be modified or freed. The data will be released by pam_end().

RETURN VALUES
In addition to the return values listed in pam(3PAM), the following value may also be returned:
PAM_NO_MODULE_DATA No module specific data is present.

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>

SEE ALSO
pam(3PAM), pam_end(3PAM), libpam(3LIB), attributes(5)
The interfaces in libpam 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   

```c
#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);
```

DESCRIPTION  The `pam_get_item()` and `pam_set_item()` functions allow applications and PAM service modules to access and to update PAM information as needed. The information is specified by `item_type`, and can be one of the following:

- **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 `pam_conv` structure.
- **PAM_USER_PROMPT**  The default prompt used by `pam_get_user()`.

For security reasons, the `item_type` `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 `pam_set_item()` function is passed the authentication handle, `pamh`, returned by `pam_start()`, a pointer to the object, `item`, and its type, `item_type`. If successful, `pam_set_item()` 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 `pam_get_item()` function is passed the authentication handle, `pamh`, returned by `pam_start()`, an `item_type`, and the address of the pointer, `item`, which is assigned the address of the requested object. The object data is valid until modified by a subsequent call to `pam_set_item()` for the same `item_type`, or unless it is modified by any of the underlying service modules. If the item has not been previously set, `pam_get_item()` returns a null pointer. An `item` retrieved by `pam_get_item()` should not be modified or freed. The item will be released by `pam_end()`.
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:

<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>

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.
### 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 `pam_sm`. The only difference between the `pam_*()` interfaces and their corresponding `pam_sm_*()` interfaces is that all the `pam_sm_*()` 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:**
  - `pam_sm_authenticate(3PAM)`
  - `pam_sm_setcred(3PAM)`

- **The second category contains the function to do account management:**
  - `pam_sm_acct_mgmt(3PAM)`

- **The third category contains the functions:**
  - `pam_sm_open_session(3PAM)`
  - `pam_sm_close_session(3PAM)`

- **The fourth category consists a function to change authentication tokens:**
  - `pam_sm_chauthtok(3PAM)`

### 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 `pam_start()`. `pam_start()` allocates space, performs various initialization
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 \texttt{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 \texttt{pam\_get\_item()} API. Though the modules can also use \texttt{pam\_set\_item()} to change any of the item information, it is recommended that nothing be changed except \texttt{PAM\_AUTHTOK} and \texttt{PAM\_OLDAUTHTOK}.

If the modules want to store any module specific state information then they can use the \texttt{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, \texttt{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 \texttt{pam\_authenticate()}, the UNIX module may store the authentication status (success or reason for failure) in the handle, using a unique name such as \texttt{SUNW\_SECURE\_RPC\_DATA}. This information is intended for use by \texttt{pam\_setcred()}. During the call to \texttt{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 \texttt{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 \texttt{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, \texttt{conv()}, along with any associated application data pointers, through the \texttt{pam\_conv} structure when it initiates an authentication transaction (via a call to \texttt{pam\_start()}). The service module will then use the function, \texttt{conv()}, to prompt the user for data, output error messages, and display text information. Refer to \texttt{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 \texttt{pam\_set\_item()} and set the value of \texttt{PAM\_USER\_PROMPT} before calling \texttt{pam\_authenticate()}. The service module’s \texttt{pam\_sm\_authenticate()} function will then call \texttt{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 \texttt{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:

- **debug**: Syslog debugging information at LOG_DEBUG level. Be careful as to not log any sensitive information such as passwords.

- **nowarn**: 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:

- **use_first_pass**: 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.

- **try_first_pass**: 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.
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.

If there are any errors, the modules should log them using syslog(3C) at the LOG_ERR level.

The PAM service module functions may return any of the PAM error numbers specified in the specific man pages. It can also return a PAM_IGNORE 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.

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>

SEE ALSO pam(3PAM), pam_authenticate(3PAM), pam_chauthtok(3PAM), pam_get_user(3PAM), pam_open_session(3PAM), pam_setcred(3PAM), pam_set_item(3PAM), pam_sm_authenticate(3PAM), pam_sm_chauthtok(3PAM), pam_sm_open_session(3PAM), pam_sm_setcred(3PAM), pam_start(3PAM), pam_strerror(3PAM), syslog(3C), pam.conf(4), attributes(5), pam_authtok_check(5), pam_authtok_get(5), pam_authtok_store(5), pam_dhkeys(5), pam_passwd_auth(5), pam_unix(5), pam_unix_account(5), pam_unix_auth(5), pam_unix_session(5)

NOTES The interfaces in libpam are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

The pam_unix(5) module might not be supported in a future release. Similar functionality is provided by pam_authtok_check(5), pam_authtok_get(5), pam_authtok_store(5), pam_dhkeys(5), pam_passwd_auth(5), pam_unix_account(5), pam_unix_auth(5), and pam_unix_session(5).
### NAME
pam_sm_acct_mgmt(3PAM)

**SYNOPSIS**
```c
#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);
```

**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, `pamh`, which is passed as the first argument to `pam_sm_acct_mgmt()`. The following flags may be set in the `flags` field:

- **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.

The `argc` argument represents the number of module options passed in from the configuration file `pam.conf(4)`. `argv` 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 options. 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, `pamh`, using `pam_set_data()`. `pam_chauthtok()` uses this information to determine which passwords have expired.

**RETURN VALUES**
If there are no restrictions to logging in, `PAM_SUCCESS` is returned. The following error values may also be returned upon error:

- **PAM_USER_UNKNOWN**: User not known to underlying authentication module.
- **PAM_NEW_AUTHTOK_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:

<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>

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.
The `pam_sm_authenticate()` 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 `pam_start()`, and is referenced by the authentication handle `pamh`.

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, `PAM_USER_UNKNOWN`.

The following flag may be passed in to `pam_sm_authenticate()`:

- **PAM_SILENT**: The 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_AUTH_ERROR**: The user has a null authentication token.

The `argc` argument represents the number of module options passed in from the configuration file `pam.conf(4)`. `argv` 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 options. If any unknown option is passed in, the module should log the error and ignore the option.

Before returning, `pam_sm_authenticate()` should call `pam_get_item()` and retrieve `PAM_AUTHTOK`. If it has not been set before and the value is `NULL`, `pam_sm_authenticate()` should set it to the password entered by the user using `pam_set_item()`.

An authentication module may save the authentication status (success or reason for failure) as state in the authentication handle using `pam_set_data(3PAM)`. This information is intended for use by `pam_setcred()`.

Upon successful completion, `PAM_SUCCESS` must be returned. In addition, the following values may be returned:

- **PAM_MAXTRIES**: Maximum number of authentication attempts exceeded.
Authentication failure.

Cannot access authentication data due to insufficient credentials.

Underlying authentication service can not retrieve authentication information.

User not known to underlying authentication module.

Ignore underlying authentication module regardless of whether the control flag is required, optional, or sufficient.

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>

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)

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.
pam_sm_chauthtok (3PAM)

NAME

pam_sm_chauthtok – service provider implementation for pam_chauthtok

SYNOPSIS

```
#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);
```

DESCRIPTION

In response to a call to `pam_chauthtok()` the PAM framework calls `pam_sm_chauthtok(3PAM)` from the modules listed in the `pam.conf(4)` file. The password management provider supplies the back-end functionality for this interface function.

The `pam_sm_chauthtok()` function changes the authentication token associated with a particular user referenced by the authentication handle `pamh`.

The following flag may be passed to `pam_chauthtok()`:

- **PAM_SILENT**
  
  The password service should not generate any messages.

- **PAM_CHANGE_EXPIRED_AUTHTOK**
  
  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_AUTHTOK**
  
  The password service should update passwords.

Note that **PAM_PRELIM_CHECK** and **PAM_UPDATE_AUTHTOK** 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 `argc` argument represents the number of module options passed in from the configuration file `pam.conf(4)`. The `argv` 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 options.

It is the responsibility of `pam_sm_chauthtok()` to determine if the new password meets certain strength requirements. `pam_sm_chauthtok()` 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.
Before returning, `pam_sm_chauthtok()` should call `pam_get_item()` and retrieve both `PAM_AUTHTOK` and `PAM_OLD_AUTHTOK`. If both are `NULL`, `pam_sm_chauthtok()` should set them to the new and old passwords as entered by the user.

**RETURN VALUES**

Upon successful completion, `PAM_SUCCESS` must be returned. The following values may also be returned:

- **PAM_PERM_DENIED** No permission.
- **PAM_AUTHTOK_ERR** Authentication token manipulation error.
- **PAM_AUTHTOK_RECOVERY_ERR** Old authentication token cannot be recovered.
- **PAM_AUTHTOK_LOCK_BUSY** Authentication token lock busy.
- **PAM_AUTHTOK_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 `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>

**SEE ALSO**

`ping(1M), pam(3PAM), pam_chauthtok(3PAM), pam_get_data(3PAM), pam_get_item(3PAM), pam_set_data(3PAM), libpam(3LIB), pam.conf(4), attributes(5)`

**NOTES**

The PAM framework invokes the password services twice. The first time the modules are invoked with the flag, `PAM_PRELIM_CHECK`. 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 `PAM_TRY_AGAIN` 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, `PAM_UPDATE_AUTHTOK`. During this stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.
If a service module receives the flag `PAM_CHANGE_EXPIRED_AUTHTok`, 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

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);

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 pamh is an authentication handle. The following flag may be set in the
flags field:

PAM_SILENT Session service should not generate any messages.

The argc argument represents the number of module options passed in from the
configuration file pam.conf(4). argv 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:

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 required, optional or sufficient.

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>

SEE ALSO
pam(3PAM), pam_open_session(3PAM), syslog(3C), libpam(3LIB), pam.conf(4),
attributes(5)
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
cce [ 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);

DESCRIPTION
In response to a call to pam_setcred(3PAM), the PAM framework calls
pam_sm_setcred() from the modules listed in the pam.conf(4) file. The
authentication provider supplies the back-end functionality for this interface function.

The pam_sm_setcred() function is called to set the credentials of the current user
associated with the authentication handle, pamh. The following flags may be set in the
flags field. Note that the first four flags are mutually exclusive:

- **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

If no flag is set, **PAM_ESTABLISH_CRED** is used as the default.

The argc argument represents the number of module options passed in from the
configuration file pam.conf(4). argv 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 **PAM_SILENT** flag is not set, then pam_sm_setcred() should print any failure
status from the corresponding pam_sm_authenticate() 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 pam_get_data(), and used to determine if user credentials should
be set.

RETURN VALUES
Upon successful completion, **PAM_SUCCESS** should be returned. The following values
may also be returned upon error:

- **PAM_CRED_UNAVAIL**: Underlying authentication service can not retrieve user credentials.
- **PAM_CRED_EXPIRED**: User credentials have expired.
pam_sm_setcred(3PAM)

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAM_USER_UNKNOWN</td>
<td>User unknown to the authentication service.</td>
</tr>
<tr>
<td>PAM_CRED_ERR</td>
<td>Failure in setting user credentials.</td>
</tr>
<tr>
<td>PAM_IGNORE</td>
<td>Ignore underlying authentication module regardless of whether the control flag is required, optional, or sufficient.</td>
</tr>
</tbody>
</table>

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>

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.
The `pam_start()` function is called to initiate an authentication transaction. `pam_start()` takes as arguments the name of the current service, `service`, the name of the user to be authenticated, `user`, the address of the conversation structure, `pam_conv`, and the address of a variable to be assigned the authentication handle `pamh`. Upon successful completion, `pamh` refers to a PAM handle for use with subsequent calls to the authentication library.

The `pam_conv` 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 `pam_conv` structure has the following entries:

```
struct pam_conv {
    int (*conv)(); /* Conversation function */
    void *appdata_ptr; /* Application data */
};
```

The `conv()` 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 `num_msg` parameter is the number of messages associated with the call. The parameter `msg` is a pointer to an array of length `num_msg` of the `pam_message` structure.

The `pam_message` 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 `pam_message` has to be allocated and freed by the PAM modules. The `pam_message` structure has the following entries:

```
struct pam_message{
    int  msg_style;
    char *msg;
};
```

The message style, `msg_style`, can be set to one of the following values:

- `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_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:

<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>
The interfaces in `libpam` are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

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.
pam_strerror(3PAM)

NAME  pam_strerror – get PAM error message string

SYNOPSIS  
```
#include <security/pam_appl.h>

const char *pam_strerror(pam_handle_t*pamh, int errnum);
```

DESCRIPTION  The pam_strerror() function maps the PAM error number in errnum to a PAM error message string, and returns a pointer to that string. The application should not free or modify the string returned.

The pamh argument is the PAM handle obtained by a prior call to pam_start(). If pam_start() returns an error, a null PAM handle should be passed.

ERRORS  The pam_strerror() function returns NULL if errnum is out-of-range.

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>

SEE ALSO  pam(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
pathfind – search for named file in named directories

SYNOPSIS
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *pathfind(const char *path, const char *name, const char *mode);

DESCRIPTION
The pathfind() function searches the directories named in path for the file name. The directories named in path are separated by colons (:). The mode argument is a string of option letters chosen from the set [rwxfbcdpugks]:

<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 name begins with a slash, it is treated as an absolute path name, and path is ignored.

An empty path member is treated as the current directory. A slash (/) character is not prepended at the occurrence of the first match; rather, the unadorned name is returned.

EXAMPLES

EXAMPLE 1 Example of finding the ls command using the PATH environment variable.

To find the ls command using the PATH environment variable:

pathfind (getenv("PATH"), "ls", "rx")

RETURN VALUES
The pathfind() function returns a (char *) value containing static, thread-specific data that will be overwritten upon the next call from the same thread.
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:

<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>

**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.
NAME  pctx_capture, pctx_create, pctx_run, pctx_release – process context library

SYNOPSIS

```
cc [ flag... ] file... -lpctx [ library... ]

#include <libpctx.h>

typedef void (*pctx_errfn_t)(const char *fn, const char *fmt, va_list ap);

pctx_t *pctx_create(const char *filename, char *const *argv, void *arg, int verbose, pctx_errfn_t *errfn);

pctx_t *pctx_capture(pid_t pid, void *arg, int verbose, pctx_errfn_t *errfn);

int pctx_run(pctx_t *pctx, uint_t sample, uint_t nsamples, int (*tick)(pctx *, pid_t, id_t, void *));

void pctx_release(pctx_t *pctx);
```

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.

There are two ways a process can be acquired by the process context functions. First, a named application can be invoked with the usual argv[] 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, arg; 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 errfn; 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.

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 sample milliseconds the process is stopped and the tick() routine is called so that, for example, the performance counters can be sampled by the caller. No periodic sampling is performed if sample is 0.
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:

<table>
<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`fork(2), cpc(3CPC), pctx_set_events(3CPC), proc(4), attributes(5)`.
pctx_set_events – associate callbacks with process events

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, ...);

**DESCRIPTION**

The `pctx_set_events()` function allows the caller (the controlling process) to express interest in various events in the controlled process. See `pctx_capture(3CPC)` for information about how the controlling process is able to create, capture and manipulate the controlled process.

The `pctx_set_events()` function takes a `pctx_t` handle, followed by a variable length list of pairs of `pctx_event_t` tags and their corresponding handlers, terminated by a `PCTX_NULL_EVENT` tag.

Most of the events correspond closely to various classes of system calls, though two additional pseudo-events (`init_lwp` and `fini_lwp`) are provided to allow callers to perform various housekeeping tasks. The `init_lwp` handler is called as soon as the library identifies a new LWP, while `fini_lwp` 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.

<table>
<thead>
<tr>
<th>System Call</th>
<th>Handler</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>exec, execve</td>
<td>fini_lwp</td>
<td>Invoked serially on all lwps in the process.</td>
</tr>
<tr>
<td></td>
<td>exec</td>
<td>Only invoked if the exec() system call failed.</td>
</tr>
<tr>
<td></td>
<td>init_lwp</td>
<td>If the exec succeeds, only invoked on lwp 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the exec fails, invoked serially on all lwps in the process.</td>
</tr>
<tr>
<td>fork, vfork,</td>
<td>fork</td>
<td>Only invoked if the fork() system call failed.</td>
</tr>
<tr>
<td>fork1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exit</td>
<td>fini_lwp</td>
<td>Invoked on all lwps in the process.</td>
</tr>
<tr>
<td></td>
<td>exit</td>
<td>Invoked on the exiting lwp.</td>
</tr>
<tr>
<td>_lwp_create</td>
<td>init_lwp</td>
<td>Only if the corresponding _lwp_create() system call succeeded.</td>
</tr>
<tr>
<td></td>
<td>lwp_create</td>
<td></td>
</tr>
<tr>
<td>_lwp_exit</td>
<td>fini_lwp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lwp_exit</td>
<td></td>
</tr>
</tbody>
</table>

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).

```c
static void
HandleExec(pctx_t *pctx, pid_t pid, id_t lwpid, char *cmd, void *arg)
{
    (void) printf("pid %d execed '%s'
", (int)pid, cmd);
```
EXAMPLE 1 HandleExec example.  (Continued)

```c
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:

<table>
<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)</td>
</tr>
<tr>
<td></td>
<td>SUNWcpcux (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
</tbody>
</table>

SEE ALSO  exec(2), exit(2), fork(2), vfork(2), fork1(2), _lwp_create(2), _lwp_exit(2), cpc(3CPC), proc(4), attributes(5).
picld_log(3PICLTREE)

NAME  picld_log – log a message in system log

SYNOPSIS  cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

void picld_log(const char *msg);

DESCRIPTION  The picld_log() function logs the message specified in msg to the system log file using syslog(3C). 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 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>

SEE ALSO  syslog(3C), attributes(5)
The `picld_plugin_register()` 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 `regp` argument.

```c
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;
```

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 `PICLD_PLUGIN_NON_CRITICAL` by plug-in modules that are not critical to system operation, and is set to `PICLD_PLUGIN_CRITICAL` by plug-in modules that are critical to the system operation. An environment control plug-in module is an example for a `PICLD_PLUGIN_CRITICAL` type of plug-in module.

The PICL daemon saves the information passed during registration in `regp` in the order in which the plug-ins registered.

Upon initialization, the PICL daemon invokes the `plugin_init()` routine of each of the registered plug-in modules in the order in which they registered. In their `plugin_init()` 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 `plugin_fini()` routines of the registered plug-in modules in the reverse order of registration. Then, the `plugin_init()` entry points are invoked again in the order in which the plug-ins registered.

Upon successful completion, 0 is returned. On failure, a negative value is returned.

**ERRORS**
- `PICL_NOTSUPPORTED` Version not supported
- `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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
picld_plugin_register(3PICLTREE)

SEE ALSO  libpicltree(3PICLTREE), attributes(5)
NAME
picl_get_first_prop, picl_get_next_prop – get a property handle of a node

SYNOPSIS
#include <picl.h>

int picl_get_first_prop(picl_nodehdl_t nodeh, piclprop_hdl_t *proph);

int picl_get_next_prop(picl_prophdl_t proph, picl_prophdl_t *nextprop);

DESCRIPTION
The picl_get_first_prop() function gets the handle of the first property of the
node specified by nodeh and copies it into the location given by proph.

The picl_get_next_prop() function gets the handle of the next property after the
one specified by proph from the property list of the node, and copies it into the location
specified by nextprop.

If there are no more properties, this function returns PICL_ENDOFLIST.

RETURN VALUES
Upon successful completion, 0 is returned. On failure, a non-negative integer is
returned to indicate an error.

PICL_ENDOFLIST is returned to indicate that there are no more properties.

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
PICL_NOTINITIALIZED Session not initialized
PICL_NORESPONSE Daemon not responding
PICL_NOTNODE Not a node
PICL_NOTPROP Not a property
PICL_INVALIDHANDLE Invalid handle
PICL_STALEHANDLE Stale handle
PICL_FAILURE General system failure
PICL_ENDOFLIST End of list

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>

Extended Library Functions 397
picl_get_first_prop(3PICL)

SEE ALSO  

picl_get_prop_by_name(3PICL), attributes(5)
NAME
picl_get_next_by_row, picl_get_next_by_col – access a table property

SYNOPSIS
cc [flag ...] file ... -lpicl [library ...]
#include <picl.h>

int picl_get_next_by_row(picl_prophdl_t proph, picl_prophdl_t *colh);
int picl_get_next_by_col(picl_prophdl_t proph, picl_prophdl_t *colh);

DESCRIPTION
The picl_get_next_by_row() function copies the handle of the property that is in
the next column of the table and on the same row as the property proph. The handle is
copied into the location given by rowh.

The picl_get_next_by_col() function copies the handle of the property that is in
the next row of the table and on the same column as the property proph. The handle is
copied into the location given by colh.

If there are no more rows or columns, this function returns the value
PICL_ENDOFLIST.

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. 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
PICL_NOTINITIALIZED  Session not initialized
PICL_NORESPONSE    Daemon not responding
PICL_NOTTABLE   Not a table
PICL_INVALIDHANDLE Invalid handle
PICL_STALEHANDLE Stale handle
PICL_FAILURE General system failure
PICL_ENDOFLIST 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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
picl_get_propval(3PICL), attributes(5)
picl_get_prop_by_name(3PICL)

NAME    picl_get_prop_by_name — get the handle of the property by name

SYNOPSIS
    cc [flag ...] file ... -lpicl [library ...]
    #include <picl.h>
    int picl_get_prop_by_name(picl_nodehdl_t nodeh, char *name,
                               picl_prophdl_t *proph);

DESCRIPTION
    The picl_get_prop_by_name() function gets the handle of the property of node
    nodeh whose name is specified in name. The handle is copied into the location specified
    by proph.

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
    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 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>

SEE ALSO
    attributes(5)
The `picl_get_propinfo()` function gets the information about the property specified by handle `proph` and copies it into the location specified by `pinfo`. The property information includes the property type, access mode, size, and the name of the property as described on `libpicl(3PICL)` manual page.

The maximum size of a property value is specified by `PICL_PROPSIZE_MAX`. It is currently set to 512KB.

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. 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:

- `PICL_NOTINITIALIZED` Session not initialized
- `PICL_NORESPONSE` Daemon not responding
- `PICL_NOTPROP` Not a property
- `PICL_INVALIDHANDLE` Invalid handle specified
- `PICL_STALEHANDLE` Stale handle specified
- `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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

See also:

- `libpicl(3PICL)`, `picl_get_propval(3PICL)`, `picl_get_propval_by_name(3PICL)`, `attributes(5)`
**NAME**
picl_get_propinfo_by_name - get property information and handle of named property

**SYNOPSIS**
```
c #include <picl.h>
int picl_get_propinfo_by_name(picl_nodehdl_t nodeh, const char *pname, picl_propinfo_t *pinfo, picl_prophdl_t *proph);
```

**DESCRIPTION**
The `picl_get_propinfo_by_name()` function copies the property information of the property specified by `pname` in the node `nodeh` into the location given by `pinfo`. The handle of the property is returned in the location `proph`.

**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**
- `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 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>

**SEE ALSO**
picl_get_propinfo(3PICL), picl_get_prop_by_name(3PICL), attributes(5)
**NAME**
picl_get_propval, picl_get_propval_by_name – get the value of a property

**SYNOPSIS**
```
#include <picl.h>

int picl_get_propval(picl_prophdl_t proph, void *valbuf, size_t nbytes);

int picl_get_propval_by_name(picl_nodehdl_t nodeh, char *propname, void *valbuf, size_t nbytes);
```

**DESCRIPTION**
The `picl_get_propval()` function copies the value of the property specified by the handle `proph` into the buffer location given by `valbuf`. The size of the buffer `valbuf` in bytes is specified in `nbytes`.

The `picl_get_propval_by_name()` function gets the value of property named `propname` of the node specified by handle `nodeh`. The value is copied into the buffer location given by `valbuf`. The size of the buffer `valbuf` in bytes is specified in `nbytes`.

The `picl_get_propval_by_name()` function is used to get a reserved property’s value. An example of a reserved property is "_parent". Please refer to `libpicl(3PICL)` 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**
- `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
PICAL_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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libpicl(3PICL), picl_get_propinfo(3PICL), attributes(5)
NAME
picl_get_root – get the root handle of the PICL tree

SYNOPSIS
cc [flag...] file ... -lpicl [library...]
#include <picl.h>

int picl_get_root(picl_nodehdl_t *nodehandle);

DESCRIPTION
The picl_get_root() function gets the handle of the root node of the PICL tree and
copies it into the location given by nodehandle.

RETURN VALUES
Upon successful completion, 0 is returned. On failure, a non-negative integer is
returned to indicate an error.

ERRORS
PICL_NOTINITIALIZED  Session not initialized
PICL_NORESPONSE     Daemon not responding
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
picl_initialize(3PICL), picl_shutdown(3PICL), attributes(5)
picl_initialize(3PICL)

NAME  picl_initialize – initiate a session with the PICL daemon

SYNOPSIS  cc [flag ...] file ... -lpicl [library ...]
           #include <picl.h>
           int picl_initialize(void);

DESCRIPTION  The picl_initialize() 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, PICL_FAILURE.

ERRORS  PICL_NOTSUPPORTED  Version not supported
        PICL_FAILURE  General system failure
        PICL_NORESPONSE  Daemon not responding

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>

SEE ALSO  picl_shutdown(3PICL), attributes(5)
NAME
picl_set_propval, picl_set_propval_by_name – set the value of a property to the specified value

SYNOPSIS
cc [flag ...] file ... -lpicl [library ...]
#include <picl.h>

int picl_set_propval(picl_prophdl_t proph, void *valbuf, size_t nbytes);

int picl_set_propval_by_name(picl_nodehdl_t nodeh, const char *pname, void *valbuf, size_t nbytes);

DESCRIPTION
The picl_set_propval() function sets the value of the property specified by the handle proph to the value contained in the buffer valbuf. The argument nbytes specifies the size of the buffer valbuf.

The picl_set_propval_by_name() function sets the value of the property named pname of the node specified by the handle nodeh to the value contained in the buffer valbuf. The argument nbytes specifies the size of the buffer valbuf.

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
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

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
picl_set_propval(3PICL)

<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>

SEE ALSO attributes(5)
NAME     picl_shutdown – shutdown the session with the PICL daemon

SYNOPSIS  cc [flag ...] file ... -lpicl [library ...]
           #include <picl.h>
           void picl_shutdown(void);

DESCRIPTION The picl_shutdown() function terminates the session with the PICL daemon and
               frees up any resources allocated.

RETURN VALUES The picl_shutdown() function does not return a value.

ERRORS      PICL_NOTINITIALIZED    Session not initialized
              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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO      picl_initialize(3PICL), attributes(5)
The `picl_strerror()` function maps the error number in `errnum` to an error message string, and returns a pointer to that string. The returned string should not be overwritten.

The `picl_strerror()` function returns `NULL` if `errnum` is out-of-range.

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>

See also `libpicl(3PICL)`, `attributes(5)`
The `picl_wait()` function blocks the calling thread until the PICL tree is refreshed. The `to_secs` argument specifies the timeout for the call in number of seconds. A value of −1 for `to_secs` specifies no timeout.

The `picl_wait()` function returns 0 to indicate that PICL tree has refreshed. Otherwise, a non-negative integer is returned to indicate error.

**ERRORS**
- `PICL_NOTINITIALIZED` Session not initialized
- `PICL_NORESPONSE` Daemon not responding
- `PICL_TIMEDOUT` Timed out waiting for refresh
- `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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**SEE ALSO** `attributes(5)`
picl_walk_tree_by_class(3PICL)

NAME
picl_walk_tree_by_class – walk subtree by class

SYNOPSIS
cc [flag ...] file ... -lpicl [library ...]
#include <picl.h>

int picl_walk_tree_by_class(picl_nodehdl_t rooth, const char *
classname, void *c_args, int (*callback)(picl_nodehdl_t nodeh, void
*c_args));

DESCRIPTION
The picl_walk_tree_by_class() function visits all the nodes of the subtree
under the node specified by rooth. The PICL class name of the visited node is
compared with the class name specified by classname. If the class names match, then
the callback function specified by callback is called with the matching node handle and
the argument provided in c_args. If the class name specified in classname 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 PICL_WALK_CONTINUE or
PICL_WALK_TERMINATE 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.

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
restared. When this occurs a client must revalidate any saved handles.

ERRORS
PICL_NOTINITIALIZED Session not initialized
PICL_NORESPONSE Daemon not responding
PICL_NOTNODE Not a node
PICL_INVALIDHANDLE Invalid handle specified
PICL_STALEHANDLE Stale handle specified
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
picl_get_propval_by_name(3PICL), attributes(5)

412 man pages section 3: Extended Library Functions • Last Revised 1 Aug 2000
pool_associate(3POOL)

NAME
pool_associate, pool_create, pool_destroy, pool_dissociate, pool_info,
pool_query_pool_resources – resource pool manipulation functions

SYNOPSIS
cc [ flag... ] file... -lpool [ library... ]
#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);

DESCRIPTION
These functions provide mechanisms for constructing and modifying pools entries
within a target pools configuration. The conf argument for each function refers to
the target configuration to which the operation applies.

The pool_associate() function associates the specified resource with pool. 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 pool_create() function creates a new pool with the supplied name with its
default properties initialized, and associated with the default resource of each type.

The pool_destroy function() destroys the given pool association. Associated
resources are not modified.

The pool_dissociate() 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 pool_info() function returns a string describing the given pool. 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 associated resources
of the pool.

The pool_query_pool_resources() function returns a null-terminated array of
resources currently associated with the pool. The return value must be freed by the
caller. The nelem argument is set to be the length of the array returned.

RETURN VALUES
Upon successful completion, pool_create() returns a new initialized pool.
Otherwise it returns NULL and pool_error(3POOL) returns the pool-specific error
value.
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.

The pool_create() function will fail if:

POE_BADPARAM   The supplied configuration’s status is not POF_VALID
or name 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:
The supplied configuration’s status is not \texttt{POF\_VALID} or the \texttt{flags} parameter is neither 0 or 1.

The configuration is invalid.

A system error has occurred. Check the system error code for more details.

The \texttt{pool\_query\_pool\_resources()} function will fail if:

- \texttt{POE\_BADPARAM} The supplied configuration’s status is not \texttt{POF\_VALID}.
- \texttt{POE\_INVALID\_CONF} The configuration is invalid.
- \texttt{POE\_SYSTEM} A system error has occurred. Check the system error code for more details.

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.

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>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>

See also \texttt{libpool(3LIB), pool\_error(3POOL), attributes(5)}
NAME  pool_component_info, pool_get_owning_resource – resource pool component functions

SYNOPSIS  cc [ flag... ] file... -lpool [ library...]  
#include <pool.h>

const char *pool_component_info(pool_conf_t *conf,  
               pool_component_t *component, int flags);

pool_resource_t *pool_get_owning_resource(pool_conf_t *conf,  
               pool_component_t *component);

DESCRIPTION  Certain resources, such as processor sets, are composed of resource components.  
Informational and ownership attributes of resource components are made available  
with the pool_component_info() and pool_get_owning_resource() functions. The conf argument for each function refers to the target configuration to  
which the operation applies.

The pool_component_info() function returns a string describing component. The  
string is allocated with malloc(3C). The caller is responsible for freeing the returned  
string. The flags argument is ignored.

The pool_get_owning_resource() function returns the resource currently  
containing component. Every component is contained by a resource.

RETURN VALUES  Upon successful completion, pool_component_info() returns a string. Otherwise  
it returns NULL and pool_error(3POOL) returns the pool-specific error value.

Upon successful completion, pool_get_owning_resource() returns the owning  
resource. Otherwise it returns NULL and pool_error() returns the pool-specific  
error value.

ERRORS  The pool_component_info() function will fail if:

POE_BADPARAM  The supplied configuration’s status is not POF_VALID  
or the flags 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_get_owning_resource() function will fail if:

POE_BADPARAM  The supplied configuration’s status is not POF_VALID.

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>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
### pool_component_info(3POOL)

**See Also**

```plaintext
libpool(3LIB), pool_error(3POOL), attributes(5)
```
NAME
    pool_component_to_elem(3POOL), pool_to_elem, pool_conf_to_elem, pool_resource_to_elem –
    resource pool element-related functions

SYNOPSIS
    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);

DESCRIPTION
    A pool element, as represented by a pool_elem_t, is a common abstraction for any
    libpool entity that contains properties. All such types can be converted to the
    opaque pool_elem_t type using the appropriate conversion functions prototyped
    above. The conf argument for each function refers to the target configuration to which
    the operation applies.

RETURN VALUES
    Upon successful completion, these functions return a pool_elem_t corresponding to
    the argument passed in. Otherwise they return NULL and pool_error(3POOL)
    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 attributes(5) 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>

SEE ALSO
    libpool(3LIB), pool_error(3POOL), attributes(5)
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 pool_conf_alloc() and released with pool_conf_free() when the configuration is no longer of interest. The conf argument for each function refers to the target configuration to which the operation applies.

The pool_conf_close() function closes the given configuration, releasing associated resources.

The pool_conf_commit() function commits changes made to the given pool_conf_t to permanent storage. If the active flag is non-zero, the state of the system will be configured to match that described in the supplied pool_conf_t. If configuring the system fails, pool_conf_commit() will attempt to restore the system to its previous state.

The pool_conf_export() function saves the given configuration to the specified location. The only currently supported value of format is POX_NATIVE, which is the format native to libpool, the output of which can be used as input to pool_conf_open().
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:

- **PO_RDONLY**
  - Open for reading only.
- **PO_RDWR**
  - Open read-write.
- **PO_CREAT**
  - Create a configuration at the given location if it does not exist. If it does, truncate it.
- **PO_DISCO**
  - Perform ‘discovery’. This option only makes sense when used in conjunction with **PO_CREAT**, and causes the returned `pool_conf_t` to contain the resources and components currently active on the system.
- **PO_UPDATE**
  - Use when opening the dynamic state file, which is the configuration at `pool_dynamic_location(3POOL)` 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:

- **POF_INVALID**
  - The configuration is not in a suitable state for use.
- **POF_VALID**
  - 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:

- **POV_LOOSE**
  - Performs basic internal syntax validation.
- **POV_STRICT**
  - Performs a more thorough syntax validation and internal consistency checks.
- **POV_RUNTIME**
  - Performs an estimate of whether attempting to commit the given configuration on the system would succeed
or fail. It is optimistic 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.

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`.

The `pool_conf_alloc()` function will fail if:

- **POE_SYSTEM**: There is not enough memory available to allocate the configuration. Check `errno` for the specific system error code.
- **POE_INVALID_CONF**: The configuration is invalid.

The `pool_conf_close()` function will fail if:

- **POE_BADPARAM**: The supplied configuration’s status is not `POF_VALID`.
- **POE_SYSTEM**: The configuration’s permanent store cannot be closed. Check `errno` for the specific system error code.

The `pool_conf_commit()` function will fail if:

- **POE_BADPARAM**: The supplied configuration’s status is not `POF_VALID` or the active flag is non-zero and the system could not be modified.
- **POE_SYSTEM**: The permanent store could not be updated. Check `errno` for the specific system error code.
- **POE_INVALID_CONF**: The configuration is not valid for this system.
- **POE_NOTSUP**: The configuration was not opened for update.
- **POE_DATASTORE**: The update of the permanent store has failed and the contents could be corrupted. Check for a `.bak` file at the datastore location if manual recovery is required.

The `pool_conf_export()` function will fail if:

- **POE_BADPARAM**: The supplied configuration’s status is not `POF_VALID` or the requested export format is not supported.
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 `flags` is neither 0 nor 1.
- **POE_SYSTEM**: There is not enough memory available to allocate the buffer used to build the information string. Check `errno` 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 `errno` for the specific system error code.
- **POE_INVALID_CONF**: The configuration to be opened is at `pool_dynamic_location(3POOL)` 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 `errno` 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 `errno` 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.
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\n', input_location);
}
```

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>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>

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

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);

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.

```
#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\n", location);
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
pool_dynamic_location(3POOL)

<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>

SEE ALSO libpool(3LIB), pool_error(3POOL), attributes(5)
**pool_error(3POOL)**

**NAME**

pool_error, pool_strerror – error interface to resource pools library

**SYNOPSIS**

```c
#include <pool.h>

int pool_error(void);

const char *pool_strerror(int perr);
```

**DESCRIPTION**

The `pool_error()` function returns the error value of the last failure recorded by the invocation of one of the functions of the resource pool configuration library, `libpool`.

The `pool_strerror()` function returns a descriptive null-terminated string for each of the valid pool error codes.

The following error codes can be returned by `pool_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; `errno(3C)` is preserved where possible.

The `pool_strerror()` function returns a pointer to the string corresponding to the requested error value. If the error value has no corresponding string, −1 is returned and `errno` is set to indicate the error.

**ERRORS**

The `pool_strerror()` function will fail if:

- **ESRCH**: The specified error value is not defined by the pools error facility.

**ATTRIBUTES**

See attributes(5) for descriptions of the following attributes:
### pool_error(3POOL)

<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>

**SEE ALSO**  
errno(3C), libpool(3LIB), pool_error(3POOL), attributes(5)
The pool_get_binding() 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), NULL is returned and the pool error value is set to POE_INVALID_SEARCH.

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 pset_() 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 pool_set_binding() function binds the processes matching idtype and id to the resources associated with pool 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 idtype parameter can be of the following types:

- P_PID The id parameter is a pid.
- P_TASKID The id parameter is a taskid.
- P_PROJID The id parameter is a project ID. All currently running processes belonging to the given project will be bound to the pool’s resources.

The pool_get_resource_binding() function returns the name of the resource of the specified type to which the process is bound. Otherwise it returns NULL and pool_error() returns the pool-specific error value.

428 man pages section 3: Extended Library Functions • Last Revised 20 Dec 2001
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.

**EXAMPLE 1** Bind the current process to the pool named "target".

```c
#include <sys/types.h>
#include <pool.h>
#include <unistd.h>

... id_t pid = getpid();
...

if (pool_set_binding("target", P_PID, pid) == POE_FAIL) {{
    (void) fprintf(stderr, "pool binding failed (\%d)\n", pool_error());
}
```

**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>CSI</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
pool_get_binding(3POOL)

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  libpool(3LIB), pool_error(3POOL), attributes(5)
### NAME
pool_get_pool, pool_get_resource, pool_query_components, pool_query_pools, pool_query_resources – retrieve resource pool configuration elements

### SYNOPSIS
```
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);
```

### DESCRIPTION
These functions provide a means for querying the contents of the specified configuration. The `conf` argument for each function refers to the target configuration to which the operation applies.

The `pool_get_pool()` function returns the pool with the given name from the provided configuration.

The `pool_get_resource()` function returns the resource with the given name and type from the provided configuration.

The `pool_query_components()` 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 `nelem`. The value returned by `pool_query_components()` is allocated with `malloc(3C)` and must be explicitly freed.

The `pool_query_pools()` function behaves similarly to `pool_query_components()` and returns the list of pools that match the given list of properties. The value returned must be freed by the caller.

The `pool_query_resources()` 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 `pool_get_pool()` and `pool_get_resource()` functions return the matching pool and resource, respectively. Otherwise, they return -1 and `pool_error(3POOL)` returns the pool-specific error value.

The `pool_query_components()`, `pool_query_pools()`, and `pool_query_resources()` 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 `pool_error()` returns the pool-specific error value.

### ERRORS
The `pool_get_pool()` 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:

<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>

SEE ALSO

libpool(3LIB), pool_error(3POOL), attributes(5)
The various pool types are converted to the common pool element type (pool_elem_t) before property manipulation. A pool_value_t is an opaque type that contains a property value of one of the following types:

- **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

The conf argument for each function refers to the target configuration to which the operation applies.

The pool_get_property() 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 POC_INVAL is returned to indicate error. Otherwise the type of the value retrieved is returned.

The pool_put_property() 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 pool_rm_property() function attempts to remove the named property from the element. If the property does not exist or is not removable, -1 is returned and pool_error(3POOL) reports an error of POE_PUTPROP.
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_Inval` 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.
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:

<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>

**SEE ALSO**

libpool(3LIB), pool_error(3POOL), attributes(5)
NAME

cpool_resource_create, pool_resource_destroy, pool_resource_info,

pool_query_resource_components, pool_resource_transfer, pool_resource_xtransfer –

resource pool resource manipulation functions

SYNOPSIS

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);

DESCRIPTION

The pool_resource_create() function creates and returns a new resource of the
given name and type in the provided configuration. If there is already a resource of the
given name, the operation will fail.

The pool_resource_destroy() function removes the specified resource from its
configuration file.

The pool_resource_info() function returns a string describing the given resource.
The string is allocated with malloc(3C). The caller is responsible for freeing the
returned string. If the flags argument is non-zero, the string returned also describes the
components (if any) contained in the resource.

The pool_query_resource_components() function returns a null-terminated
array of the components (if any) that comprise the given resource.

The pool_resource_transfer() function transfers size basic units from the source
resource to the target. 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 size, while increasing the size property of the
target by the same.

The pool_resource_xtransfer() function transfers the specific components from
the source resource to the target. Both resources must be of the same type, and of a type
that contains components (such as processor sets). The components argument is a
null-terminated list of pool_component_t.
The conf 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 name 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 flags parameter 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`.  

Extended Library Functions
pool_resource_create(3POOL)

<table>
<thead>
<tr>
<th>POE_INVALID_CONF</th>
<th>The configuration is invalid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE_SYSTEM</td>
<td>A system error has occurred. Check the system error code for more details.</td>
</tr>
</tbody>
</table>

The `pool_resource_transfer()` function will fail if:

<table>
<thead>
<tr>
<th>POE_BADPARAM</th>
<th>The supplied configuration’s status is not <code>POF_VALID</code>, the two resources are not of the same type, or the transfer would cause either of the resources to exceed their <code>min</code> and <code>max</code> properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE_SYSTEM</td>
<td>A system error has occurred. Check the system error code for more details.</td>
</tr>
</tbody>
</table>

The `pool_resource_xtransfer()` function will fail if:

<table>
<thead>
<tr>
<th>POE_BADPARAM</th>
<th>The supplied configuration’s status is not <code>POF_VALID</code>, the two resources are not of the same type, or the supplied resources do not belong to the source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE_INVALID_CONF</td>
<td>The transfer operation failed and the configuration may be invalid.</td>
</tr>
<tr>
<td>POE_SYSTEM</td>
<td>A system error has occurred. Check the system error code for more details.</td>
</tr>
</tbody>
</table>

EXAMPLES

**EXAMPLE 1** Create a new resource of type `pset` named `foo`.

```c
#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\n");
    ...
}
```

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>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>
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 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);

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
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 value does not match the type of the requested operation.

---

**RETURN VALUES**

Upon successful completion, `pool_value_alloc()` returns a pool property value with type initialized to PVC_INVAL. 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, PVC_INVAL 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()` 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 value 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:

<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>

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
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 *));

DESCRIPTION
The walker functions provided with libpool(3POOL) visit each associated entity of the given type, and call the caller-provided callback function with a user-provided additional opaque argument. There is no implied order of visiting nodes in the walk. If the callback function returns a non-zero value at any of the nodes, the walk is terminated, and an error value of -1 returned. The conf argument for each function refers to the target configuration to which the operation applies.

The pool_walk_components() function invokes callback on all components contained in the resource.

The pool_walk_pools() function invokes callback on all pools defined in the configuration.

The pool_walk_resources() function invokes callback function on all resources associated with pool.

RETURN VALUES
Upon successful completion of the walk, these functions return 0. Otherwise -1 is returned and pool_error(3POOL) returns the pool-specific error value.

ERRORS
These functions 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.

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>CSI</td>
<td>Enabled</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
</tbody>
</table>
pool_walk_components(3POOL)

<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>

SEE ALSO  libpool(3LIB), pool_error(3POOL), attributes(5)
The pow() function computes the value of x raised to the power y, \(x^y\). If x is negative, y must be an integer value.

Upon successful completion, pow() 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 errno may be set to EDOM or ERANGE.

If the correct value would cause overflow, +HUGE_VAL is returned, and errno is set to ERANGE.

If the correct value would cause underflow to 0, 0 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.

The pow() function will fail if:

- **EDOM** The value of x is negative and y is non-integral.
- **ERANGE** The value to be returned would have caused overflow.

The pow() function may fail if:

- **EDOM** The value of x is 0.0 and y is negative.
- **ERANGE** The correct value would cause underflow.

An application wishing to check for error situations should set errno to 0 before calling pow(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

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>

See also exp(3M), isnan(3M), matherr(3M), attributes(5), standards(5)
printDmiAttributeValues(3DMI)

NAME
printDmiAttributeValues – print data in input DmiAttributeValues list

SYNOPSIS
c [ flag ... ] file ... -ldmi -linsl -lrwtool [ library ... ]
#include <dmi/util.hh>

void printDmiAttributeValues(DmiAttributeValues_t *values);

DESCRIPTION
The printDmiAttributeValues() function prints the data in the input DmiAttributeValues list. The function prints "unknown data" for those values that contain invalid data.

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>

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

NAME
printDmiDataUnion – print data in input data union

SYNOPSIS
ce [ 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:

<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>

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

NAME
printDmiString – print a DmiString

SYNOPSIS
e [ 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:

<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>

SEE ALSO
newDmiString(3DMI), libdmi(3LIB), attributes(5)
project – access project files from Perl

SYNOPSIS

use Sun::Solaris::Project qw(:ALL);

my ($name, $projid, $comment, $users, $groups, $attr) = getprojent();
($name, $projid, $comment, $users, $groups, $attr) = getprojbyname("proj");
my $proj = getdefaultproj("root");

DESCRIPTION

This module provides perl access to the project file library as documented in getprojent(3PROJECT). The interface is similar to the standard perl getxxx() functions such as getpwent() and gethostent(). For detailed descriptions of the individual functions, refer to the getprojent(3PROJECT) and project(4) manual pages.

CONSTANTS

PROJNAME_MAX  maximum length of a project name

FUNCTIONS

getprojent()  
Returns the next entry from the projects file. When called in a scalar context, getprojent() returns just the name of the project, or undef when the end of the file is reached. When called in a list context, getprojent() returns a 6-element list consisting of ($name, $projid, $comment, $users, $groups, $attr). $users and $groups are returned as references to arrays containing the appropriate user or project lists. On end of file, undef is returned.

setprojent()  
Rewinds the project database to the beginning of the file.

derprojent()  
Closes the project file.

getprojid()  
Returns the current numeric project ID.

getprojbyname($name)  
Searches the project database for an entry with the specified name, returning undef if it cannot be found or a 6-element list as returned by getprojent() if it can be found.

getprojbyid($id)  
Searches the project database for an entry with the specified ID, returning undef if it cannot be found or a 6-element list as returned by getprojent() if it can be found.

getdefaultproj($user)  
Returns the default project entry for the specified user in the same format as getprojent(), or undef if the user cannot be found. For full details of the lookup process, see the manual page for getdefaultproj(3PROJECT).

fgetprojent($filehandle)  
Returns the next project entry from $filehandle, which is a perl file handle, and must refer to a previously opened file in project(4) format. Return values are the same as for getprojent().
inproj($user, $project)
Checks to see if the specified user is able to use the project. Returns TRUE if the
user can use the project and FALSE otherwise.

getprojidbyname($project)
Searches the project database for the specified project and returns the project ID if it
is found. If not found, undef is returned.

EXPORTS
By default nothing is exported from this namespace. The following tags can be used to
selectively import constants and functions defined in this namespace:

:LIBCALLS PROJNAME_MAX, getprojent(), setprojent(),
endprojent(), getprojbyname(), getprojbyid(),
getdefaultproj(), fgetprojent(), inproj(),
getprojidbyname(), getprojid() :LIBCALLS

:ALL :LIBCALLS

EXAMPLES
EXAMPLE 1 Get the record for the default project and print its list of attributes.

use Sun::Solaris::Project qw(:ALL);
my ($name, $projid, $comment, $users, $groups, $attr) =
getprojbyname("default");
die("Can't find default project\n") if (! defined($name));
print("Project $name:\n");
print(" Project id: $projid\n");
print(" Comment: $comment\n");
print(" Users: @$users\n");
print(" Groups: @$groups\n");
print(" Attributes: $attr\n");

SEE ALSO perl(1), getdefaultproj(3PROJECT), getprojent(3PROJECT), project(4)
project_walk(3PROJ)

NAME
project_walk – visit active project IDs on current system

SYNOPSIS
cc [ flag ... ] file ... -lproject [ library ... ]
#include <project.h>

int project_walk(int (*callback)(const projid_t project, void *
*walk_data), void *init_data);

DESCRIPTION
The project_walk() function provides a mechanism for the application author to
examine all active projects on the current system. The callback function provided by the
application is given the ID of an active project at each invocation and can use the
walk_data 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, project_walk() returns 0. It returns −1 if the callback
function returned a non-zero value or if the walk encountered an error, in which case(errno is set to indicate the error.

ERRORS
The project_walk() function will fail if:
ENOMEM There is insufficient memory available to set up the initial data for
the walk.

Other returned error values are presumably caused by the callback function.

EXAMPLES
EXAMPLE 1 Count the number of projects available on the system.

The following example counts the number of projects available on the system.

#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;
ero = 0;
if (n=project_walk(simple_callback, &total_count)) >= 0)
   (void) printf("count = %u\n", total_count.count);
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>

SEE ALSO getprojid(2), settaskid(2), attributes(5)
NAME
ptree_add_node, ptree_delete_node – add or delete node to or from tree

SYNOPSIS
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_add_node(picl_nodehdl_t parh, picl_nodehdl_t chdh);
int ptree_delete_node(ptree_delete_node nodeh);

DESCRIPTION
The ptree_add_node() function adds the node specified by handle chdh as a child node to the node specified by the handle parh. PICL_CANTPARENT is if the child node already has a parent.

The ptree_delete_node() function deletes the node specified by handle nodeh 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.

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_NOTNODE Node a node
PICL_CANTPARENT Already has a parent
PICL_TREEBUSY PICL tree is busy
PICL_INVALIDHANDLE Invalid handle
PICL_STALEHANDLE Stale handle
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
ptree_add_prop(3PICLTREE)

NAME  
ptree_add_prop, ptree_delete_prop – add or delete a property

SYNOPSIS  
cc [flag...] file... -lpicltree [library...]
#include <picltree.h>

int ptree_add_prop(picl_nodehdl_t nodeh, picl_prophdl_t proph);
int proph(picl_prophdl_t proph);

DESCRIPTION  
The ptree_add_prop() function adds the property specified by the handle proph to the list of properties of the node specified by handle nodeh.

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  
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_create_and_add_node(picl_nodehdl_t parh, const char *name, const char *classname, picl_nodehdl_t *nodeh);

DESCRIPTION
The ptree_create_and_add_node() function creates a node with the name and PICL class specified by name and classname respectively. It then adds the node as a child to the node specified by parh. The handle of the new node is returned in nodeh.

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_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 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>

SEE ALSO
ptree_create_node(3PICLTREE), ptree_add_node(3PICLTREE), attributes(5)
ptree_create_and_add_prop(3PICLTREE)

NAME  ptree_create_and_add_prop – create and add property to node and return property handle

SYNOPSIS  cc [flag...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_create_and_add_prop(picl_nodehdl_t nodeh,
               ptree_propinfo_t *infop, void *vbuf, picl_prophdl_t *proph);

DESCRIPTION  The ptree_create_and_add_prop() function creates a property using the
property information specified in infop and the value buffer vbuf and adds the property
to the node specified by nodeh. If proph is not NULL, the handle of the property added
to the node is returned in proph.

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_PROPEXIST      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>
<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>

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
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_create_node(char *name, char *clname, picl_nodehdl_t *nodeh);

int ptree_destroy_node(picl_nodehdl_t nodeh);

DESCRIPTION
The ptree_create_node() function creates a node and sets the "name" property value to the string specified in name and the "class" property value to the string specified in clname. The handle of the new node is copied into the location given by nodeh.

The ptree_destroy_node() function destroys the node specified by nodeh and frees up any allocated space. The node to be destroyed must have been previously deleted by ptree_delete_node (see ptree_add_node(3PICLTREE)). Otherwise, PICL_CANTDESTROY is returned.

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_INVALIDARG Invalid argument
PICL_VALUETOObIG Value exceeds maximum size
PICL NOT SUPPORTED Property version not supported
PICL CANTDESTROY Attempting to destroy before delete
PICL TREETBUSY PICL tree is busy
PICL NOTNODE Not a node
PICL INVALIDHANDLE Invalid handle
PICL STALEHANDLE Stale handle
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ptree_add_node(3PICLTREE), attributes(5)

Extended Library Functions 457
ptree_create_prop(3PICLTREE)

NAME  ptree_create_prop, ptree_destroy_prop — create or destroy a property

SYNOPSIS  cc [flag...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_create_prop(ptree_propinfo_t *pinfo, void *valbuf,
                       picl_prophdl_t *proph);

int ptree_destroy_prop(pici_prophdl_t proph);

DESCRIPTION  The ptree_create_prop() function creates a property using the information
speciﬁed in pinfo, 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 valbuf, which may be NULL for volatile properties.
The handle of the property created is copied into the location given by proph. See
libpicltree(3PICLTREE) for more information on the structure of
ptree_propinfo_t structure.

The ptree_destroy_prop() function destroys the property specified by the handle
proph. 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  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 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>
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  cc [flag ...] file ... -lpicltree [library ...]
           #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);

DESCRIPTION  The ptree_create_table() function creates a table object and returns the handle of the table in tbl_hdl.

The ptree_add_row_to_table() function adds a row of properties to the table specified by tbl_hdl. The handles of the properties of the row are specified in the proph array and nprops 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  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 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>

SEE ALSO  attributes(5)
NAME
ptree_find_node – find node with given property and value

SYNOPSIS
cc [flag...] file ... -lpicltree [library...]
#include <picltree.h>

int ptree_find_node(picl_nodehdl_t rooth, char *pname,
                    picl_prop_type_t ptype, void *pval, size_t valsize,
                    picl_nodehdl_t *retnodeh);

DESCRIPTION
The ptree_find_node() function visits the nodes in the subtree under the node
specified by rooth. The handle of the node that has the property whose name, type,
and value matches the name, type, and value specified in pname, ptype, and pval
respectively, is returned in the location given by retnodeh. The argument valsize gives
the size of the value in pval. The first valsize number of bytes of the property value is
compared with pval.

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
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 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>

SEE ALSO
ptree_get_prop_by_name(3PICLTREE), ptree_get_propinfo(3PICLTREE),
ptree_get_propval(3PICLTREE), ptree_get_propval_by_name(3PICLTREE),
atributes(5)
ptree_get_first_prop(3PICLTREE)

NAME
ptree_get_first_prop, ptree_get_next_prop – get a property handle of the node

SYNOPSIS
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_get_first_prop (picl_nodehdl_t nodeh, picl_prophdl_t *proph);
int ptree_get_next_prop (picl_prophdl_t proph, picl_prophdl_t *nextproph);

DESCRIPTION
The ptree_get_first_prop() function gets the handle of the first property of the
node specified by nodeh and copies it into the location specified by proph.

The ptree_get_next_prop() function gets the handle of the next property after
the one specified by proph from the list of properties of the node and copies it into the
location specified by nextproph.

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_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 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>

SEE ALSO
ptree_get_prop_by_name(3PICLTREE), attributes(5)
NAME
ptree_get_next_by_row, ptree_get_next_by_col – access a table property

SYNOPSIS
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);

DESCRIPTION
The ptree_get_next_by_row() function copies the handle of the property that is in the next column of the table and on the same row as the property proph. The handle is copied into the location given by rowh.

The ptree_get_next_by_col() function copies the handle of the property that is in the next row of the table and on the same column as the property proph. The handle is copied into the location given by colh.

If there are no more rows or columns, this function returns the value PICL_ENDOFLIST.

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_NOTTABLE Not a table
PICL_INVALIDHANDLE Invalid handle
PICL_STALEHANDLE Stale handle
PICL_ENDOFLIST End of list
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ptree_create_table(3PICLTREE), attributes(5)
ptree_get_node_by_path

NAME  ptree_get_node_by_path – get handle of node specified by PICL tree path

SYNOPSIS

cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_get_node_by_path(const char *ptreepath, picl_nodehdl_t *nodeh);

DESCRIPTION

The ptree_get_node_by_path() function copies the handle of the node in the
PICL tree specified by the path given in ptreepath into the location nodeh.

The syntax of a PICL tree path is:

[def_propname:] / [def_propval [match_cond] ... ]

where def_propname prefix is a shorthand notation to specify the name of the property
whose value is specified in def_propval, and the match_cond expression specifies the
matching criteria for that node in the form of one or more pairs of property names and
values such as

[address] [?prop_name = prop_val] ... ]

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 prop_name gives the
property name and prop_val gives the property value for non PICL_PTYPE_VOID
properties. The values for properties of type PICL_PTYPE_TABLE,
PICL_PTYPE_BYTEARRAY, and PICL_PTYPE_REFERENCE cannot be specified in the
match_cond 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.

PICL_NOTNODE is returned if there is no node corresponding to the specified path.

ERRORS

PICL_INVALIDARG  Invalid argument
PICL_NOTNODE    Not a node
PICL_FAILURE    General system failure

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
ptree_get_node_by_path(3PLICLTREE)

<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>

SEE ALSO ptree_get_propval_by_name(3PLICLTREE), attributes(5)
ptree_get_prop_by_name(3PICLTREE)

NAME
   ptree_get_prop_by_name – get a property handle by name

SYNOPSIS
   cc [flag ...] file ... -lpicltree [library ...]
   #include <picltree.h>
   int ptree_get_prop_by_name(picl_nodehdl_t nodeh,
                               char *name,
                               picl_prophdl_t *proph);

DESCRIPTION
   The ptree_get_prop_by_name() function gets the handle of the property, whose
   name is specified in name, of the node specified by the handle nodeh. The property
   handle is copied into the location specified by proph.

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
   ptree_get_propval_by_name(3PICLTREE) 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
   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 attributes(5) for descriptions of the following attributes:

   +---------------------------------+------------------+
   | ATTRIBUTE TYPE | ATTRIBUTE VALUE  |
   +----------------+------------------+
   | MT-Level        | MT-Safe          |
   +----------------+------------------+

SEE ALSO
   ptree_get_first_prop(3PICLTREE),
   ptree_get_propval_by_name(3PICLTREE), attributes(5)
ptree_get_propinfo(3PICLTREE)

NAME ptree_get_propinfo – get property information

SYNOPSIS

cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_get_propinfo(picl_prophdl_t proph, ptree_propinfo_t *pi);

DESCRIPTION

The ptree_get_propinfo() function gets the information about the property specified by handle proph and copies it into the location specified by pi. See libpicltree(3PICLTREE) for more information about ptree_propinfo_t 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

PICL_INVALIDHANDLE Invalid handle
PICL_STALEHANDLE Stale handle
PICL_NOTPROP Not a property
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

libpicltree(3PICLTREE), ptree_create_prop(3PICLTREE), attributes(5)
ptree_get_propinfo_by_name(3PICLTREE)

NAME
ptree_get_propinfo_by_name – get property information and handle of named property

SYNOPSIS
cc [flag...] file... -lpicltree [library...]
#include <picltree.h>

int ptree_get_propinfo_by_name(picl_nodehdl_t nodeh, const char *pname, ptree_propinfo_t *pinfo, picl_prophdl_t *proph);

DESCRIPTION
The ptree_get_propinfo_by_name() function copies the property information of the property specified by *pname in the node nodeh into the location given by pinfo. The handle of the property is returned in the location proph.

RETURN VALUES
Upon successful completion, 0 is returned. On failure, a non-negative integer is returned to indicate an error.

ERRORS
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 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>

SEE ALSO
ptcl_get_propinfo(3PICLTREE), ptcl_get_prop_by_name(3PICLTREE), attributes(5)
**ptree_get_propval(3PICLTREE)**

NAME
ptree_get_propval, ptree_get_propval_by_name – get the value of a property

SYNOPSIS
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);

DESCRIPTION
The ptree_get_propval() function gets the value of the property specified by the handle proph and copies it into the buffer specified by valbuf. The size of the buffer valbuf is specified in nbytes.

The ptree_get_propval_by_name() function gets the value of the property, whose name is specified by name, from the node specified by handle nodeh. The value is copied into the buffer specified by valbuf. The size of the buffer is specified by nbytes.

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
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 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>

SEE ALSO
ptree_update_propval(3PICLTREE), attributes(5)
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:

<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>

SEE ALSO libpicltree(3PICLTREE), ptree_create_node(3PICLTREE), attributes(5)
ptree_init_propinfo(3PICLTREE)

NAME
ptree_init_propinfo – initialize ptree_propinfo_t structure

SYNOPSIS
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_init_propinfo(ptree_propinfo_t *infop, int version, int ptype, int pmode, size_t psize, char *pname, int (*readfn)(ptree_rarg_t *, void *), int (*writefn)(ptree_warg_t *, const void *));

DESCRIPTION
The ptree_init_propinfo() function initializes a ptree_propinfo_t property information structure given by location infop with the values provided by the arguments.

The version argument specifies the version of the ptree_propinfo_t structure. PTREE_PROPINFO_VERSION gives the current version. The arguments ptype, pmode, psize, and pname specify the property’s PICL type, access mode, size, and name. The maximum size of a property name is defined by PICL_PROPNAMELEN_MAX. The arguments readfn and writefn specify a volatile property’s read and write access functions. For non-volatile properties, these are set to NULL.

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_NOTSUPPORTED Property version not supported
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ptree_get_propinfo(3PICLTREE), attributes(5)
The `ptree_post_event()` function posts the specified event and its arguments to the PICL framework. The argument `ename` specifies a pointer to a string containing the name of the PICL event. The arguments `earg` and `size` specify a pointer to a buffer containing the event arguments and size of that buffer, respectively. The argument `completion_handler` 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 `ename`, `earg`, and `size` 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.

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**

- PICL_INVALIDARG: Invalid argument
- 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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

**See Also**

- `ptree_register_handler(3PICLTREE)`,
- `ptree_unregister_handler(3PICLTREE)`,
- `attributes(5)`
NAME
ptree_register_handler – register a handler for the event

SYNOPSIS
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);

DESCRIPTION
The ptree_register_handler() function registers an event handler for a PICL event. The argument ename specifies the name of the PICL event for which to register the handler. The argument evt_handler specifies the event handler function. The argument cookie 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
void evt_handler(const char *ename, const void *earg, \size_t size, void *cookie)

where, ename, earg, size, and cookie are the arguments passed to the event handler when it is invoked. The argument ename is the PICL event name for which the handler is invoked. The arguments earg and size gives the pointer to the event argument buffer and its size, respectively. The argument cookie is the pointer to the caller specific data registered with the handler. The arguments ename and earg 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
PICL_INVALIDARG    Invalid argument
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ptree_unregister_handler(3PICLTREE), attributes(5)
ptree_unregister_handler(3PICLTREE)

NAME
ptree_unregister_handler – unregister the event handler for the event

SYNOPSIS
cc [flag ...] file ... -lpicltree [library ...]
#include <picltree.h>

void ptree_unregister_handler(const char *ename, void (*evt_handler)(const char *ename, const void *earg, size_t size, void *cookie), void *cookie);

DESCRIPTION
The ptree_unregister_handler() function unregisters the event handler for the specified event. The argument ename specifies the name of the PICL event for which to unregister the handler. The argument evt_handler specifies the event handler function. The argument cookie 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 ptree_unregister_handler() or a deadlock may result.

The ptree_unregister_handler() function must not be invoked from the handler that is being unregistered.

RETURN VALUES
This function does not return a value.

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>

SEE ALSO
ptree_register_handler(3PICLTREE), attributes(5)
ptree_update_propval(3PICLTREE)

NAME
ptree_update_propval, ptree_update_propval_by_name – update a property value

SYNOPSIS
cc [flags...] file ... -lpicltree [library ...]
#include <picltree.h>

int ptree_update_propval(picl_prophdl_t proph, void *valbuf, size_t nbytes);

int ptree_update_propval_by_name(picl_nodehdl_t nodeh, char *name, void *valbuf, size_t nbytes);

DESCRIPTION
The ptree_update_propval() function updates the value of the property specified by proph with the value specified in the buffer valbuf. The size of the buffer valbuf is specified in nbytes.

The ptree_update_propval_by_name() function updates the value of the property, whose name is specified by name, of the node specified by handle nodeh. The new value is specified in the buffer valbuf, whose size is specified in nbytes.

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
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 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>

SEE ALSO
ptree_get_propval(3PICLTREE), attributes(5)
ptree_walk_tree_by_class(3PICLTREE)

NAME  ptree_walk_tree_by_class – walk subtree by class

SYNOPSIS

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));

DESCRIPTION

The ptree_walk_tree_by_class() function visits all the nodes of the subtree
under the node specified by rooth. The PICL class name of the visited node is
compared with the class name specified by classname. If the class names match, the
callback function specified by callback is called with the matching node handle and the
argument provided in c_args. If the class name specified in classname 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 PICL_WALK_CONTINUE or
PICL_WALK_TERMINATE 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

PICL_NOTNODE          Not a node
PICL_INVALIDHANDLE    Invalid handle specified
PICL_STALEHANDLE      Stale handle specified
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>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

ptree_get_propval_by_name(3PICLTREE), attributes(5)
read_vtoc(3EXT)

NAME
read_vtoc, write_vtoc – read and write a disk’s VTOC

SYNOPSIS
cc [ flag ... ] file ... -ladm [ library ... ]
#include <sys/vtoc.h>

int read_vtoc(int fd, struct vtoc *vtoc);
int write_vtoc(int fd, struct vtoc *vtoc);

DESCRIPTION
The read_vtoc() function returns the VTOC (volume table of contents) structure
that is stored on the disk associated with the open file descriptor fd.

The write_vtoc() function stores the VTOC structure on the disk associated with
the open file descriptor fd.

The fd argument refers to any slice on a raw disk.

RETURN VALUES
Upon successful completion, read_vtoc() returns a positive number indicating the
slice index associated with the open file descriptor. Otherwise, it returns a negative
number indicating one of the following errors:

VT_EIO An I/O error occurred.
VT_ERROR An unknown error occurred.

Upon successful completion, write_vtoc() returns 0. Otherwise, it returns a
negative number indicating one of the following errors:

VT_EIO An I/O error occurred.
VT_ERROR An unknown error occurred.
VT_EINVAL The VTOC contains an incorrect field.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
fmthard(1M), format(1M), prtvtoc(1M), ioctl(2), attributes(5), dkio(7I)

BUGS
The write_vtoc() function cannot write a VTOC on an unlabeled disk. Use
format(1M) for this purpose.
reg_ci_callback(3DMI)

NAME  reg_ci_callback – provide a component instrumentation with a transient program number

SYNOPSIS  
```
#include <dmi/ci_callback_svc.hh>

u_long reg_ci_callback();
```

DESCRIPTION  The reg_ci_callback() function provides a component instrumentation with a transient program number. The instrumentation uses this number to register its RPC service provider. The prognum member of the DmiRegisterInfo structure is populated with the return value of this function.

RETURN VALUES  Upon successful completion, the reg_ci_callback() function returns a transient program number of type u_long.

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>Unafe</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5)
NAME

regexpr, compile, step, advance – regular expression compile and match routines

SYNOPSIS

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[];

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 ed(1).

The parameter instring is a null-terminated string representing the regular expression.

The parameter expbuf points to the place where the compiled regular expression is to be placed. If expbuf is NULL, compile() uses malloc(3C) 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 endbuf is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if expbuf is NULL. If the compiled expression cannot fit in (endbuf–expbuf) bytes, compile() returns NULL and regerrno (see below) is set to 50.

The parameter string is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The parameter expbuf is the compiled regular expression obtained by a call of the function compile().

The function step() 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 step(). The variables set in step() are loc1 and loc2. loc1 is a pointer to the first character that matched the regular expression. The variable loc2 points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, loc1 points to the first character of string and loc2 points to the null at the end of string.

The purpose of step() is to step through the string argument until a match is found or until the end of string is reached. If the regular expression begins with ^, step() tries to match the regular expression at the beginning of the string only.
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.

### EXAMPLE 1

The following is similar to the regular expression code from `grep`:

```c
#include<regex.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:

<table>
<thead>
<tr>
<th>ERROR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Range endpoint too large.</td>
</tr>
<tr>
<td>16</td>
<td>Bad Number.</td>
</tr>
<tr>
<td>ERROR</td>
<td>MEANING</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>25</td>
<td>&quot;\digit&quot; out or range.</td>
</tr>
<tr>
<td>36</td>
<td>Illegal or missing delimiter.</td>
</tr>
<tr>
<td>41</td>
<td>No remembered string search.</td>
</tr>
<tr>
<td>42</td>
<td>(\sim) imbalance.</td>
</tr>
<tr>
<td>43</td>
<td>Too many (.</td>
</tr>
<tr>
<td>44</td>
<td>More than 2 numbers given in (}&amp;\sim).</td>
</tr>
<tr>
<td>45</td>
<td>) expected after (.</td>
</tr>
<tr>
<td>46</td>
<td>First number exceeds second in (\sim).</td>
</tr>
<tr>
<td>49</td>
<td>[ ] imbalance.</td>
</tr>
<tr>
<td>50</td>
<td>Regular expression overflow.</td>
</tr>
</tbody>
</table>

**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>

**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.
NAME
remainder – remainder function

SYNOPSIS
#include <math.h>

double remainder(double x, double y);

DESCRIPTION
The remainder() 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 \( |n - x/y| \leq \frac{1}{2} \), the value \( n \) is chosen to be even.

The behavior of remainder() is independent of the rounding mode.

RETURN VALUES
The remainder() function returns the floating point remainder \( r = x - ny \) when \( y \) is non-zero.

When \( y \) is 0, remainder() returns NaN and sets errno to EDOM.

If the value of \( x \) is ±Inf, remainder() returns NaN and sets errno to EDOM.

If \( x \) or \( y \) is NaN, then the function returns NaN.

ERRORS
The remainder() function will fail if:

EDOM The \( y \) argument is 0 or the \( x \) argument is positive or negative infinity.

USAGE
The remainder() function computes the remainder \( x \ REM y \) required by ANSI/IEEE 754 (IEC 559).

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>

SEE ALSO
fmod(3M), attributes(5)
NAME
rint – round-to-nearest integral value

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double rint(double x);

DESCRIPTION
The rint() 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 rint() is
identical to floor(3M). If the current rounding mode rounds toward positive infinity,
then rint() is identical to ceil(3M).

RETURN VALUES
Upon successful completion, the rint() function returns the integer (represented as a
double precision number) nearest x in the direction of the current IEEE754 rounding
mode.

When x is ±Inf, rint() returns x.

If the value of x is NaN, NaN is returned.

ERRORS
No errors will occur.

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>

SEE ALSO
ceil(3M), floor(3M), isnan(3M), attributes(5)
The `rsm_create_localmemory_handle()` and `rsm_free_localmemory_handle()` functions are supporting functions for `rsm_memseg_import_putv(3RSM)` and `rsm_memseg_import_getv(3RSM)`.

The `rsm_create_localmemory_handle()` function creates a local memory handle to be used in the I/O vector component of a scatter-gather list of subsequent `rsm_memseg_import_putv()` and `rsm_memseg_import_getv()` calls. The `handle` argument specifies the controller handle obtained from `rsm_get_controller(3RSM)`. The `l_handle` argument is a pointer to the location for the function to return the local memory handle. The `local_vaddr` argument specifies the local virtual address; it should be aligned at a page boundary. The `length` argument specifies the length of memory spanned by the handle.

The `rsm_free_localmemory_handle()` function unlocks the memory range for the local handle specified by `l_handle` and releases the associated system resources. The `handle` argument specifies the controller handle. All handles created by a process are freed when the process exits, but the process should call `rsm_free_localmemory_handle()` as soon as possible to free the system resources.

Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

The `rsm_create_localmemory_handle()` and `rsm_free_localmemory_handle()` functions can return the following errors:

- `RSMERR_BAD_CTLR_HNDL` Invalid controller handle.
- `RSMERR_BAD_LOCALMEM_HNDL` Invalid local memory handle.

The `rsm_create_localmemory_handle()` function can return the following errors:

- `RSMERR_BAD_LENGTH` Invalid length.
- `RSMERR_BAD_ADDRESS` Invalid address.
- `RSMERR_INSUFFICIENT_MEM` Insufficient memory.

See `attributes(5)` for descriptions of the following attributes:
rsm_create_localmemory_handle(3RSM)

<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>

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  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);

DESCRIPTION  The controller functions provide mechanisms for obtaining access to a controller, determining the characteristics of the controller, and releasing the controller.

The rsm_get_controller() function acquires a controller handle through the controller argument. The name argument is the specific controller instance (for example, "sci0" or "loopback"). This controller handle is used for subsequent RSMAPI calls.

The rsm_get_controller_attr() function obtains a controller’s attributes through the attr argument. The chdl argument is the controller handle obtained by the rsm_get_controller() call. The attribute structure is defined in the <rsmapi> header.

The rsm_release_controller() function releases the resources associated with the controller identified by the controller handle chdl, obtained by calling rsm_get_controller(). Each rsm_release_controller() call must have a corresponding rsm_get_controller() 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 rsm_get_controller(), rsm_get_controller_attr(), and rsm_release_controller() functions can return the following errors:

RSMERR_BAD_CTLR_HNDL  Invalid controller handle.

The rsm_get_controller() and rsm_get_controller_attr() functions can return the following errors:

RSMERR_BAD_ADDR  Bad address.

The rsm_get_controller() function can return the following errors:

RSMERR_CTLR_NOT_PRESENT  Controller not present.
RSMERR_INSUFFICIENT_MEM  Insufficient memory.
RSMERR_BAD_LIBRARY_VERSION  Invalid library version.

ATTRIBUTES  See attributes(5) for descriptions of the following attributes:
rsm_get_controller(3RSM)

<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>

SEE ALSO
rsm_memseg_export_create(3RSM), rsm_memseg_import_connect(3RSM), attributes(5)
The `rsm_get_interconnect_topology()` and `rsm_free_interconnect_topology()` 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 `<rsmapi.h>` header.

The `rsm_get_interconnect_topology()` returns a pointer to the topology data in a location specified by the `topology_data` argument.

The `rsm_free_interconnect_topology()` frees the resources allocated by `rsm_get_interconnect_topology()`.

Upon successful completion, `rsm_get_interconnect_topology()` returns 0. Otherwise, an error value is returned to indicate the error.

The `rsm_get_interconnect_topology()` function can return the following errors:

- `RSMERR_BAD_TOPOLOGY_PTR` Invalid topology pointer.
- `RSMERR_INSUFFICIENT_MEM` Insufficient memory.
- `RSMERR_BAD_ADDR` Bad address.

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>
rsm_get_interconnect_topology(3RSM)

<table>
<thead>
<tr>
<th>Attributes</th>
<th>MT-Level</th>
<th>MT-Safe</th>
</tr>
</thead>
</table>

SEE ALSO attributes(5)
rsm_get_segmentid_range(3RSM)

NAME  rsm_get_segmentid_range – get segment ID range

SYNOPSIS  cc [flags...] file... -lrsm [library...]
#include <rsmapi.h>

int rsm_get_segmentid_range(const char *appid, rsm_segment_id_t *baseid, uint_t *length);

DESCRIPTION  RSM segment IDs can be either specified by the application or generated by the system using the rsm_memseg_export_publish(3RSM) function. Applications that specify segment IDs require a reserved range of segment IDs that they can use. This can be achieved by using rsm_get_segmentid_range() and by reserving a range of segment IDs in the segment ID configuration file, /etc/rsm/rsm.segmentid. The rsm_get_segmentid_range() function can be used by applications to obtain the segment ID range reserved for them. The appid argument is a null-terminated string that identifies the application. The baseid argument points to the location where the starting segment ID of the reserved range is returned. The length argument points to the location where the number of reserved segment IDs is returned.

The application can use any value starting at baseid and less than baseid+length. The application should use an offset within the range of reserved segment IDs to obtain a segment ID such that if the baseid or length 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 /etc/rsm/rsm.segmentid file are of the form:

#keyword appid baseid length
reserve SUNWfoo 0x600000 1000

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 baseid (the starting segment ID of the reserved range in hexadecimal), and the length (the number of segment ids 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 </usr/include/rsm/rsm_common.h> 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 rsm_get_segmentid_range() function can return the following errors:

RSMERR_BAD_ADDR  The address passed is invalid.
RSMERR_BAD_APPID  The appid 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.
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>Unstable</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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

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);

DESCRIPTION

The rsm_intr_signal_post() and rsm_intr_signal_wait() 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 rsm_intr_signal_wait(). A process can signal a waiting process when an event occurs by calling rsm_intr_signal_post().

The rsm_intr_signal_post() function signals an event occurrence. Either an import segment handle (rsm_memseg_import_handle_t) or an export segment handle (rsm_memseg_export_handle_t) may be type cast to a void pointer for the memseg argument. If memseg refers to an import handle, the exporting process is signalled. If memseg refers to an export handle, all importers of that segment are signalled. The flags argument may be set to RSM_SIGPOST_NO_ACCUMULATE; this will cause this event to be discarded if an event is already pending for the target segment.

The rsm_intr_signal_wait() function allows a process to block and wait for an event occurrence. Either an import segment handle (rsm_memseg_import_handle_t) or an export segment handle (rsm_memseg_export_handle_t) may be type cast to a void pointer for the memseg argument. The process blocks for up to timeout 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 rsm_intr_signal_post() and rsm_intr_signal_wait() functions can return the following errors:

RSMERR_BAD_SEG_HNDL Invalid segment handle.

The rsm_intr_signal_post() function can return the following error:

RSMERR_REMOTE_NODE_UNREACHABLE Remote node not reachable.

The rsm_intr_signal_wait() function can return the following errors:

RSMERR_TIMEOUT Timer expired.
RSMERR_INTERRUPTED Wait interrupted.

ATTRIBUTES

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

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
rsm_intr_signal_post(3RSM)

<table>
<thead>
<tr>
<th>Interface Stability</th>
<th>Evolving</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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
cc [flags...] file... -lrsms [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);

DESCRIPTION
The rsm_memseg_export_create(), rsm_memseg_export_destroy(), and
rsm_memseg_export_rebind() 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 rsm_memseg_export_create() 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 controller argument is the controller handle obtained from a prior
call to rsm_get_controller(3RSM). The export memory segment handle is
obtained through the memseg argument for use in subsequent operations. The vaddr
argument specifies the process virtual address for the segment. It must be aligned
according to the controller page size attribute. The length argument specifies the size of
the segment in bytes and must be in multiples of the controller page size. The flags
argument is a bitmask of flags. The RSM_ALLOW_REBIND flag indicates that unbind
and rebinding is allowed on the segment during its lifetime. The RSM_LOCK_OPS flag
indicates that this segment can be used for lock operations.

The rsm_memseg_export_destroy() function deallocates the physical memory
pages associated with the segment and disconnects all importers of the segment. The
memseg argument is the export memory segment handle obtained by a call to
rsm_memseg_export_create().

The rsm_memseg_export_rebind() function releases the current backing pages
associated with the segment and associates 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 rebinding operation has completed.
Segment data access during the rebinding operation does not cause a system failure but data content
results are undefined. The memseg argument is the export segment handle pointer
obtained from rsm_memseg_export_create(). The vaddr argument must be
aligned with respect to the page size attribute of the controller. The length argument
modulo controller page size must be 0. The off 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 rsm_memseg_export_create(), rsm_memseg_export_destroy(), and rsm_memseg_export_rebind() functions can return the following errors:

- **RSMERR_BAD_SEG_HNDL** Invalid segment handle.
- **RSMERR_BAD_CTLR_HNDL** Invalid controller handle.
- **RSMERR_CTLR_NOT_PRESENT** Controller not present.
- **RSMERR_BAD_LENGTH** Length zero or length exceeds controller limits.
- **RSMERR_BAD_ADDR** Invalid address.
- **RSMERR_INSUFFICIENT_MEM** Insufficient memory.
- **RSMERR_INSUFFICIENT_RESOURCES** Insufficient resources.
- **RSMERR_PERM_DENIED** Permission denied.
- **RSMERR_NOT_CREATOR** Not creator of segment.
- **RSMERR_REBIND_NOT_ALLOWED** Rebind not allowed.

The rsm_memseg_export_create() function can return the following errors:

- **RSMERR_BAD_MEM_ALIGNMENT** The address is not aligned on a page boundary.

The rsm_memseg_export_rebind() function can return the following errors:

- **RSMERR_INTERRUPTED** The operation was interrupted by a signal.

The rsm_memseg_export_destroy() function can return the following errors:

- **RSMERR_POLLFD_IN_USE** 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 valloc() (see malloc(3)) or mmap(2). Memory for a file mapped with mmap() must be mapped MAP_PRIVATE.

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>Unstable</td>
</tr>
</tbody>
</table>
rsm_memseg_export_create(3RSM)

<table>
<thead>
<tr>
<th>MT-Level</th>
<th>MT-Safe</th>
</tr>
</thead>
</table>

SEE ALSO
rsm_get_controller(3RSM), rsm_memseg_export_publish(3RSM), attributes(5)
The `rsm_memseg_export_publish()`, `rsm_memseg_export_unpublish()`, and `rsm_memseg_export_republish()` functions allow or disallow a memory segment to be imported by other nodes.

The `rsm_memseg_export_publish(3RSM)` function allows the export segment specified by the `memseg` 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 `segment_id` 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 `access_list` 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:

- 2: write access
- 4: read only access
- 6: read and write access

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 `access_list_length` 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 `access_list` argument, the `access_list_length` argument specifies the number of entries in the `access_list` array.
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:

<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>
rsm_memseg_export_publish(3RSM)

<table>
<thead>
<tr>
<th>MT-Level</th>
<th>MT-Safe</th>
</tr>
</thead>
</table>

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  

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);

DESCRIPTION  The rsm_memseg_get_pollfd() and rsm_memseg_release_pollfd() functions provide an alternative to rsm_intr_signal_wait(3RSM); the waiting process may multiplex event waiting using the poll(2) function after first obtaining a poll descriptor using rsm_memseg_get_pollfd(). The descriptor may subsequently be released using rsm_memseg_release_pollfd().

As a result of a call rsm_memseg_get_pollfd(), the specified pollfd structure is initialized with a descriptor for the specified segment (memseg) and the event generated by rsm_inetr_signal_post(3RSM). Either an export segment handle or an import segment handle may be type cast to a void pointer. The pollfd argument may subsequently be used with the poll(2) function to wait for the event. If memseg references an export segment, the segment must be currently published. If memseg references an import segment, the segment must be connected.

The rsm_memseg_release_pollfd() function decrements the reference count of the pollfd 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 rsm_memseg_get_pollfd() and rsm_memseg_release_pollfd() function can return the following errors:

RSMERR_BAD_SEG_HNDL  Invalid segment handle.

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

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

SEE ALSO  poll(2), rsm_inetr_signal_post(3RSM), attributes(5)
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
cc [flags...] file... -l rsm [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);

DESCRIPTION
The rsm_memseg_import_connect() function provides a means of creating an import segment called memseg and establishing a logical connection with an export segment identified by the segment_id on the node specified by node_id. The controller specified by controller must have a physical connection with the controller (see rsm_get_interconnect_topology(3RSM)) used while exporting the segment identified by segment_id on node specified by node_id. The perm 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 perm argument is limited to the following octal values:

0400  read mode
0200  write mode
0600  read/write mode

The rsm_memseg_import_disconnect() function breaks the logical connection between the import segment and the exported segment and deallocates the resources associated with the import segment handle memseg.

RETURN VALUES
Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

ERRORS
The rsm_memseg_import_connect() and rsm_memseg_import_disconnect() functions can return the following errors:

RSMERR_BAD_SEG_HNDL   Invalid segment handle.

The rsm_memseg_import_connect() function can return the following errors:

RSMERR_BAD_CTLR_HNDL   Invalid controller handle.
RSMERR_CTLR_NOT_PRESENT
   Controller not present.
RSMERR_PERM_DENIED
   Permission denied.
The `rsm_memseg_import_disconnect()` function can return the following errors:

- **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 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:

<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>

**SEE ALSO**

`rsm_memseg_import_map(3RSM)`, `attributes(5)`
When using interconnects that allow memory mapping (see rsm_memseg_import_map(3RSM)), 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 rsm_memseg_import_open_barrier(3RSM)). The default barrier operation mode is RSM_BARRIER_MODE_IMPLICIT, 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 rsm_memseg_import_set_mode(3RSM)). In either case the barrier should be initialized prior to using these functions using rsm_memseg_import_init_barrier(3RSM).

The rsm_memseg_import_get() function copies length bytes from the imported segment im_memseg beginning at location offset from the start of the segment to a local memory buffer pointed to by dest_addr.

The rsm_memseg_import_get8() function copies rep_cnt number of 8-bit quantities from successive locations starting from offset in the imported segment to successive local memory locations pointed to by datap.

The rsm_memseg_import_get16() functions copies rep_cnt number of 16-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 half-word address boundary.
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 64-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.

Upon successful completion, these functions return 0. Otherwise, an error value is returned to indicate the error.

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.

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>

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

```
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);
```

DESCRIPTION

The rsm_memseg_import_init_barrier() function creates a barrier for the imported segment specified by memseg. The barrier type is specified by the type argument. Currently, only RSM_BAR_DEFAULT is supported as a barrier type. A handle to the barrier is obtained through the barrier argument and is used in subsequent barrier calls.

The rsm_memseg_import_destroy_barrier() 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 rsm_memseg_import_init_barrier() and rsm_memseg_import_destroy_barrier() functions can return the following errors:

- RSMERR_BAD_SEG_HNDL   Invalid segment handle.
- RSMERR_BAD_BARRIER_PTR Invalid barrier pointer.

The rsm_memseg_import_init_barrier() function can return the following errors:

- RSMERR_INSUFFICIENT_MEM Insufficient memory.

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>

SEE ALSO

rsm_memseg_import_open_barrier(3RSM), rsm_memseg_import_set_mode(3RSM), attributes(5)
rsm_memseg_import_map(3RSM)

NAME  rsm_memseg_import_map, rsm_memseg_import_unmap – map or unmap imported segment

SYNOPSIS  cc [flags...] file... -lrsms [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);

DESCRIPTION  The rsm_memseg_import_map() and rsm_memseg_import_unmap() 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 rsm_memseg_import_get(3RSM) and rsm_memseg_import_put(3RSM) manual pages.

The rsm_memseg_import_map() function maps an import segment into caller’s address space for the segment to be accessed by CPU memory operations. The im_memseg 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 address argument. The attr argument can be one of the following:

RSM_MAP_NONE  The system will choose available virtual address to map and return its value in the address argument.

RSM_MAP_FIXED  The import segment should be mapped at the requested virtual address specified in the address argument.

The perm argument determines whether read, write or a combination of accesses are permitted to the data being mapped. It can be either RSM_PERM_READ, RSM_PERM_WRITE, or RSM_PERM_RDWR.

The offset argument is the byte offset location from the base of the segment being mapped to address. The length argument indicates the number of bytes from offset to be mapped.

The rsm_memseg_import_unmap() 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 rsm_memseg_import_map() and rsm_memseg_import_unmap() functions can return the following errors:

RSMERR_BAD_SEG_HNDL  Invalid segment handle.
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:

<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>

**SEE ALSO**

- `rsm_memseg_import_connect(3RSM)`
- `rsm_memseg_import_get(3RSM)`
- `rsm_memseg_import_put(3RSM)`
- `rsm_memseg_get_pollfd(3RSM)`
- `attributes(5)`
NAME
rsm_memseg_import_open_barrier, rsm_memseg_import_order_barrier,
rsm_memseg_import_close_barrier – remote memory access error detection functions

SYNOPSIS
cc [flags...] file... -lrsms [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);

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:
RSMERR_BAD_SEG_HNDL Invalid segment handle
RSMERR_BAD_BARRIER_PTR Invalid barrier pointer.

The rsm_memseg_close_barrier() and rsm_memseg_order_barrier() functions can return the following errors:
RSMERR_BARRIER_UNINITIALIZED Barrier not initialized.
RSMERR_BARRIER_NOT_OPENED Barrier not opened.
RSMERR_BARRIER_FAILURE Memory access error.
RSMERR_CONN_ABORTED Connection aborted.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
rsm_memseg_import_open_barrier(3RSM)

<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>

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
cc [flags...] file... -lram [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);

DESCRIPTION
When using interconnects that allow memory mapping (see
rsm_memseg_import_map(3RSM)), 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
rsm_memseg_import_open_barrier(3RSM)). The default barrier operation mode
is RSM_BARRIER_MODE_IMPLICIT, 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 rsm_memseg_import_set_mode(3RSM)).

The rsm_memseg_import_put() function copies length bytes from local memory
with start address src_addr to the imported segment im_memseg beginning at location
offset from the start of the segment.

The rsm_memseg_import_put8() function copies rep_cnt number of 8-bit quantities
from successive local memory locations pointed to by datap to successive locations
starting from offset in the imported segment.

The rsm_memseg_import_put16() function copies rep_cnt number of 16-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
half-word address boundary.

The rsm_memseg_import_put32() function copies rep_cnt number of 32-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
word address boundary.
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:

<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>

**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)
The `rsm_memseg_import_putv()` and `rsm_memseg_import_getv()` functions provide for using a list of I/O requests rather than a single source and destination address as is done for the `rsm_memseg_import_put()` and `rsm_memseg_import_get()` functions.

The I/O vector component of the scatter-gather list (`sg_io`), allows specifying local virtual addresses or local_memory_handles. 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 `rsm_create_localmemory_handle()` and `rsm_free_localmemory_handle()`.

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 `IMPLICIT` 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 `io_request_count` field of the `rsm_scat_gath_t` pointed to by `sg_io`. The `io_request_count` is valid if greater than 0 and less than or equal to `RSM_MAX_SGIOREQS`. If `io_request_count` is not in the valid range, `rsm_memseg_import_putv()` and `rsm_memseg_import_getv()` return `RSMERR_BAD_SGIO`.

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 ever I/O transfer operation. The means of enabling the implicit signal post.
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:

- **RSMERR_BAD_SGIO**: Invalid scatter-gather structure pointer.
- **RSMERR_BAD_SEG_HNDL**: Invalid segment handle.
- **RSMERR_BAD_CTRLR_HNDL**: Invalid controller handle.
- **RSMERR_BAD_OFFSET**: Invalid offset.
- **RSMERR_BAD_LENGTH**: Invalid length.
- **RSMERR_BAD_ADDR**: Bad address.
- **RSMERR_INSUFFICIENT_RESOURCES**: Insufficient resources.
- **RSMERR_INTERRUPTED**: The operation was interrupted by a signal.
- **RSMERR_PERM_DENIED**: Permission denied.
- **RSMERR_BARRIER_FAILURE**: I/O completion error.
- **RSMERR_REMOTE_NODE_UNREACHABLE**: Remote node not reachable.

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>

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 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);

DESCRIPTION The rsm_memseg_import_set_mode() function provides support for optional explicit barrier scoping in the functions described on the rsm_memseg_import_get(3RSM) and rsm_memseg_import_put(3RSM) manual pages. The two valid barrier modes are RSM_BARRIER_MODE_EXPLICIT and RSM_BARRIER_MODE_IMPLICIT. By default, the barrier mode is set to RSM_BARRIER_MODE_IMPLICIT. When the mode is RSM_BARRIER_MODE_IMPLICIT, 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 rsm_memseg_import_init_barrier(3RSM) function before any barrier operations, either implicit or explicit, are used.

The rsm_memseg_import_get_mode() 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 rsm_memseg_import_set_mode() and rsm_memseg_import_get_mode() functions can return the following errors:

RSMERR_BAD_SEG_HNDL Invalid segment handle.

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>

SEE ALSO rsm_memseg_import_get(3RSM), rsm_memseg_import_init_barrier(3RSM), rsm_memseg_import_put(3RSM), attributes(5)
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,
laversion – runtime linker auditing functions

SYNOPSIS

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);

cchar *la_objsearch(const char *name, uintptr_t *cookie, uint_t flag);

uintptr_t la_objopen(Link_map *lm, 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_sparcv9_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);

uintptr_t la_version(uint_t version);

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 Linker and Libraries Guide for a full description of the link auditing
mechanism.

SEE ALSO

ld.so.1(1)

Linker and Libraries Guide
rtld_db(3EXT)

NAME | rtld_db, rd_delete, rd_errno, 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
c [ flag ... ] file ... -lrtld_db [ library ... ]
#include <proc_service.h>
#include <rtld_db.h>

void rd_delete(struct rd_agent *rdap);
char *rd_errno(rd_errno rderr);
rd_errno rd_event_addr(rd_agent *rdap, rd_notify_t *notify);
rd_errno rd_event_enable(struct rd_agent *rdap, int onoff);
rd_errno rd_event_getmsg(struct rd_agent *rdap, rd_event_msg_t *msg);
rd_errno rd_init(int version);
typedef int rl_iter_f(const rd_loadobj_t *, void *);
rd_errno 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_errno rd_objpad_enable(struct rd_agent *rdap, size_t padsize);
rd_errno rd_plt_resolution(rd_agent *rdap, paddr_t pc, lwpid_t lwpid, paddr_t plt_base, rd_plt_info_t *rpi);
rd_errno rd_reset(struct rd_agent *rdap);

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 Linker and Libraries
Guide for a full description of the runtime linker debugger interface mechanism.

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>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
ld.so.1(1), librtld_db(3LIB), libthread_db(3THR), attributes(5)

Linker and Libraries Guide

Last Revised 14 Aug 2001
scalb – load exponent of a radix-independent floating-point number

SYNOPSIS
#include <math.h>

double scalb(double x, double n);

DESCRIPTION
The scalb() function computes \( x \times r^n \), where \( r \) is the radix of the machine’s floating point arithmetic. When \( r \) is 2, scalb() is equivalent to ldexp(3C).

RETURN VALUES
Upon successful completion, the scalb() function returns \( x \times r^n \).

If the correct value would overflow, scalb() returns ±HUGE_VAL (according to the sign of \( x \)) and sets errno to ERANGE.

If the correct value would underflow to 0.0, scalb() returns 0 and sets errno to ERANGE.

The scalb() function returns \( x \) when \( x \) is ±Inf.

If \( x \) or \( n \) is NaN, then scalb() returns NaN.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The scalb() function will fail if:

ERANGE The correct value would overflow or underflow.

USAGE
An application wishing to check for error situations should set errno to 0 before calling scalb(). 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>
<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>

SEE ALSO ldexp(3C), matherr(3M), attributes(5)
scalbn(3M)

NAME   scalbn – load exponent of a radix-independent floating-point number

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double scalbn(double x, int n);

DESCRIPTION The scalbn() function computes \( x \times r^n \), where \( r \) is the radix of the machine’s floating point arithmetic.

RETURN VALUES Upon successful completion, the scalbn() function returns \( x \times r^n \).

If the correct value would overflow, scalbn() returns \( \pm \text{HUGE_VAL} \) (according to the sign of \( x \)).

The scalbn() function returns \( x \) when \( x \) is \( \pm \text{Inf} \).

If \( x \) is NaN, then scalbn() returns NaN.

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>

SEE ALSO attributes(5)
sendfile(3EXT)

NAME  sendfile – send files over sockets or copy files to files

SYNOPSIS  
cc [flag ...] file ... -lsendfile [library ...]
#include <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd, off_t *off, size_t len);

DESCRIPTION  The sendfile() function copies data from out_fd to in_fd starting at offset off and of length len bytes. The in_fd argument should be a file descriptor to a regular file opened for reading. See open(2). The out_fd 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 socket(3SOCKET). The off argument is a pointer to a variable holding the input file pointer position from which the data will be read. After sendfile() has completed, the variable will be set to the offset of the byte following the last byte that was read. The sendfile() function does not modify the current file pointer of in_fd, but does modify the file pointer for out_fd if it is a regular file.

The sendfile() function can also be used to send buffers by pointing in_fd to SFV_FD_SELF.

RETURN VALUES  Upon successful completion, sendfile() returns the total number of bytes written to out_fd and also updates the offset to point to the byte that follows the last byte read. Otherwise, it returns −1, and errno is set to indicate an error.

ERRORS  The sendfile() function will fail if:

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 out_fd or in_fd argument is either not a valid file descriptor, out_fd is not opened for writing, or in_fd is not opened for reading.
EINVAL  The offset cannot be represented by the off_t structure, or the length is negative when cast to ssize_t.
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 out_fd argument is no longer connected to the peer endpoint.

USAGE  The sendfile() function has a transitional interface for 64-bit file offsets. See lf64(5).
**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.

```c
int tfd;
off_t baddr;
struct sockaddr_in sin;
in_addr_t addr;

char buf[64 * 1024];

int 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("Transfered %d bytes from buffer to socket\n");
```

**EXAMPLE 2** Transferring Files to Sockets

The following program demonstrates a transfer of files to sockets:

```c
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) {
    perror("socket");
    exit(1);
}

printf("Transfered %d bytes from file to socket\n");
```
EXAMPLE 2 Transferring Files to Sockets  (Continued)

```c
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcs(32–bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcslx (64–bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

FILES  /usr/lib/libsendfile.so.1  shared object file

SEE ALSO  open(2), socket(SOCKET), attributes(5), lf64(5)
The `sendfilev()` function attempts to write data from the `sfvcnt` buffers specified by the members of `vec` array: `vec[0]`, `vec[1]`, ..., `vec[sfvcnt-1]`. `fi`ldes is a file descriptor to a regular file or to a `AF_NCA`, `AF_INET`, or `AF_INET6` family type `SOCK_STREAM` socket that is open for writing.

This function is analogous to the `writev()` system call. See `writev(2)`. However, instead of sending out chunks of data, `sendfilev()` can read input data from data buffers or file descriptors.

The following is the `sendfilevec` structure:

```c
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;
```

To send a file, open the file for reading. Point `sfv_fd` to the file descriptor returned as a result. See `open(2)`. `sfv_off` should contain the offset within the file. `sfv_len` should have the length of the file to be transferred.

The `xferred` parameter is updated to record the total number of bytes written to `out_fd`.

The `sfv_flag` field is reserved and should be set to zero.

To send data directly from the address space of the process, set `sfv_fd` to `SFV_FD_SELF`. `sfv_off` should point to the data, with `sfv_len` containing the length of the buffer.

The `sendfilev()` function supports the following parameters:

- `fi`ldes: A file descriptor to a regular file or to a `AF_NCA`, `AF_INET`, or `AF_INET6` family type `SOCK_STREAM` socket that is open for writing. For `AF_NCA`, the protocol type should be zero.
- `vec`: An array of `SENDFILEVEC_T`, as defined in the `sendfilevec` structure above.
- `sfvcnt`: The number of members in `vec`.
- `xferred`: The total number of bytes written to `out_fd`. 

522 man pages section 3: Extended Library Functions • Last Revised 25 Apr 2001
Upon successful completion, `sendfilev()` returns total number of bytes written to `out_fd`. Otherwise, it returns -1, and `errno` is set to indicate an error. `xferred` contains the amount of data successfully transferred, which can be used to discover the error vector.

**ERRORS**

- **EAFNOSUPPORT**: The implementation does not support the specified address family for socket.
- **EPROTOTYPE**: The socket type is not supported.
- **EBADF**: The `fildes` argument is not a valid descriptor open for writing or an `sfv_fd` is invalid or not open for reading.
- **EACCES**: The process does not have appropriate privileges or one of the files pointed by `sfv_fd` does not have appropriate permissions.
- **EPIPE**: The `fildes` argument is a socket that has been shut down for writing.
- **EIO**: An I/O error occurred while accessing the file system.
- **EFAULT**: The `vec` argument points to an illegal address.
- **EFAULT**: The `xferred` argument points to an illegal address.
- **EINVAL**: The `sfvcnt` argument was less than or equal to 0. One of the `sfv_len` in `vec` array was less than or equal to 0, or greater than the file size. An `sfv_fd` is not seekable.
- **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.

**USAGE**
The `sendfilev()` function has a transitional interface for 64-bit file offsets. See `lfs64(5)`.

**EXAMPLES**
The following example sends 2 vectors, one of HEADER data and a file of length 100 over `sockfd`. `sockfd` is in a connected state, that is, `socket()`, `accept()`, and `bind()` operation are complete.

```c
#include <sys/sendfile.h>

int main (int argc, char eargv[])
{
    int sockfd;
    ssize_t ret;
    size_t xfer;
    struct sendfilevec vec[2];
    ...
    ...
    ...
    return 0;
}
```
sendfilev(3EXT)

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcsl(32-bit)</td>
</tr>
<tr>
<td></td>
<td>SUNWcslx (64-bit)</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Evolving</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
open(2), writev(2), attributes(5)
setproject(3PROJECT)

NAME
setproject – place process in new project with attendant resource controls, resource pools, and attributes

SYNOPSIS
cc [ flag ... ] file... -lproject [ library ... ]
#include <project.h>

int setproject(const char *project_name, const char *user_name, uint_t flags);

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 user_name is a valid member of the project specified by project_name, as determined by inproej(3PROJECT), setproject() will create a new task with settaskid(2) using task flags specified by flags, 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 user_name is a name of the superuser (user with UID equal to 0), the setproject() function skips the inproej(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 project_name 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:

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 user_name 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:

<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>

SEE ALSO pooladm(1M), setrctl(2), settaskid(2), inprooj(3PROJECT),
pool_error(3POOL), pool_set_binding(3POOL), passwd(4), project(4),
attributes(5)
significand function

cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double significand(double x);

The significand() function, along with the logb(3M) and scalb(3M) 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 \( \text{sign} \times 2^n \) with 1 < sign < 2, then significand(x) returns sign for exercising the fraction-part(F) test vector. significand(x) is not defined when x is either 0, \( \pm \text{Inf} \) or NaN.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by various Standards.

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>

See also logb(3M), matherr(3M), scalb(3M), attributes(5)
sin(3M)

NAME  

sin – sine function

SYNOPSIS  

cc [ flag ... ] file ... -lm [ library ... ]  
#include <math.h>

double sin(double x);

DESCRIPTION  

The sin() function computes the sine of its argument x, measured in radians.

RETURN VALUES  

Upon successful completion, sin() returns the sine of x.

If x is NaN or ±Inf, NaN is returned.

ERRORS  

No errors will occur.

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>

SEE ALSO  

asin(3M), isnan(3M), attributes(5)
NAME
sinh – hyperbolic sine function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double sinh(double x);

DESCRIPTION
The sinh() function computes the hyperbolic sine of x.

RETURN VALUES
Upon successful completion, sinh() returns the hyperbolic sine of x.

If the result would cause an overflow, ±HUGE_VAL is returned and errno is set to ERANGE.

If x is NaN, NaN is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS
The sinh() function will fail if:

ERANGE The result would cause overflow.

USAGE
An application wishing to check for error situations should set errno to 0 before calling sinh(). 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>
<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>

SEE ALSO
asinh(3M), cosh(3M), isnan(3M), matherr(3M), tanh(3M), attributes(5), standards(5)
sqrt(3M)

NAME   sqrt – square root function

SYNOPSIS cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double sqrt(double x);

DESCRIPTION The sqrt() function computes the square root of x.

RETURN VALUES Upon successful completion, sqrt() returns the square root of x.

If x is NaN, NaN is returned.
If x is negative, NaN is returned and errno is set to EDOM.

ERRORS The sqrt() function will fail if:
EDOM The value of x is negative.

USAGE An application wishing to check for error situations should set errno to 0 before calling sqrt(). 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>
<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>

SEE ALSO isnan(3M), attributes(5)
SSAgentIsAlive(3SNMP)

NAME
SSAgentIsAlive, SSAGetTrapPort, SSARegSubtable, SSARegSubagent,
SSARegSubtree, SSASendTrap, SSASubagentOpen – Sun Solstice Enterprise Agent
registration and communication helper functions

SYNOPSIS
cc [ flag ... ] file ... -lssagent -lssasnmp [ library .. ]
#include <impl.h>

extern int SSAgentIsAlive(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);

DESCRIPTION
The SSAgentIsAlive() function returns TRUE if the master agent is alive,
otherwise returns FALSE. The agent_addr parameter is the address of the agent. Specify
the security token in the community parameter. You can specify the maximum amount
of time to wait for a response with the timeout 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 agent parameter is a pointer to an Agent structure containing the
following members:

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 */

The agent_id member is an integer value returned by the SSASubagentOpen() function. After calling SSASubagentOpen(), you pass the agent_id in the
SSARegSubagent() call to register the subagent with the Master Agent.

The following values are supported for agent_status:
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.
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:

<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>

**SEE ALSO** attributes(5)
SSAOidCmp(3SNMP)

NAME
SSAOidCmp, SSAOidCpy, SSAOidDup, SSAOidFree, SSAOidInit, SSAOidNew, SSAOidString, SSAOidStrToOid, SSAOidZero – Sun Solstice Enterprise Agent OID helper functions

SYNOPSIS
cc [ flag ... ] file ... -lssasnmp [ 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);

DESCRIPTION
The SSAOidCmp() function performs a comparison of the given OIDs. This function returns:

0 if oid1 is equal to oid2
1 if oid1 is greater than oid2
-1 if oid1 is less than oid2

The SSAOidCpy() function makes a deep copy of oid2 to oid1. This function assumes oid1 has been processed by the SSAOidZero() function. Memory is allocated inside oid1 and the contents of oid2, not just the pointer, is copied to oid1. If an error is encountered, an error message is stored in the error_label buffer.

The SSAOidDup() function returns a clone of oid, by using the deep copy. Error information is stored in the error_label 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 subids to the OID instance with the specified length len. 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 error_label buffer.

The SSAOidString() function returns a char pointer for the printable form of the given oid.
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:

<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>

SEE ALSO

attributes(5)
SSAStringCpy(3SNMP)

NAME  SSAStringCpy, SSAStringInit, SSAStringToChar, SSAStringZero – Sun Solstice Enterprise Agent string helper functions

SYNOPSIS
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);

DESCRIPTION

The SSAStringCpy() function makes a deep copy of string2 to string1. This function assumes that string1 has been processed by the SSAStringZero() function. Memory is allocated inside the string1 and the contents of string2, not just the pointer, is copied to the string1. If an error is encountered, an error message is stored in the error_label buffer.

The SSAStringInit() function copies the char array from chars to the string instance with the specified length len. 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 error_label 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>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)
NAME
strcpy, stradd, strcadd, strecpy – copy strings, compressing or expanding escape codes

SYNOPSIS
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *strccpy(char *output, const char *input);
char *strcadd(char *output, const char *input);
char *strecpy(char *output, const char *input, const char *exceptions);
char *streadd(char *output, const char *input, const char *exceptions);

DESCRIPTION
strccpy() copies the input string, up to a null byte, to the output string, compressing the C-language escape sequences (for example, \n, \001) to the equivalent character. A null byte is appended to the output. The output argument must point to a space big enough to accommodate the result. If it is as big as the space pointed to by input it is guaranteed to be big enough. strccpy() returns the output argument.

strcadd() is identical to strccpy(), except that it returns the pointer to the null byte that terminates the output.

strecpy() copies the input string, up to a null byte, to the output string, expanding non-graphic characters to their equivalent C-language escape sequences (for example, \n, \001). The output argument must point to a space big enough to accommodate the result; four times the space pointed to by input is guaranteed to be big enough (each character could become \ and 3 digits). Characters in the exceptions string are not expanded. The exceptions argument may be zero, meaning all non-graphic characters are expanded. strecpy() returns the output 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.
/* 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);

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

+--------------------------------+---------------------+
| ATTRIBUTE TYPE | ATTRIBUTE VALUE     |
+----------------+---------------------+
| MT-Level        | MT-Safe             |
+--------------------------------+---------------------+

SEE ALSO
string(3C), strfind(3GEN), attributes(5)
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.
NAME  strfind, strrspn, strtrns, str – string manipulations

SYNOPSIS  

cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int strfind(const char *as1, const char *as2);
char *strrspn(const char *string, const char *tc);
char *strtrns(const char *string, const char *old, const char *new,
              char *result);

DESCRIPTION  The strfind() function returns the offset of the first occurrence of the second string, as2, if it is a substring of string as1. If the second string is not a substring of the first string strfind() returns −1.

The strrspn() function trims characters from a string. It searches from the end of string for the first character that is not contained in tc. If such a character is found, strrspn() returns a pointer to the next character; otherwise, it returns a pointer to string.

The strtrns() function transforms string and copies it into result. Any character that appears in old is replaced with the character in the same position in new. The new result is returned.

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

EXAMPLES  

EXAMPLE 1 An example of the strfind() function.

/* find offset to substring "hello" within as1 */
i = strfind(as1, "hello");
/* trim junk from end of string */
s2 = strrspn(s1, "*?#$");
*s2 = '\0';
/* transform lower case to upper case */
a1[] = "abcdefghijklmnopqrstuvwxyz";
a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
s2 = strtrns(s1, a1, a2, s2);

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>

SEE ALSO  string(3C), attributes(5)
### NAME
sysevent_bind_handle, sysevent_unbind_handle – bind or unbind subscriber handle

### SYNOPSIS
cc [flag...] file ... -lsysevent [library...]
#include <libsysevent.h>

```c
sysevent_handle_t *sysevent_bind_handle(void
    (*event_handler)(sysevent_t *ev), void *cookie);
```

```c
void sysevent_unbind_handle(sysevent_handle_t *sysevent_hdl);
```

### PARAMETERS
- `ev` pointer to sysevent buffer handle
- `event_handler` pointer to an event handling function
- `sysevent_hdl` pointer to a sysevent subscriber handle

### DESCRIPTION
The `sysevent_bind_handle()` function allocates memory associated with a subscription handle and binds it to the caller’s `event_handler`. The `event_handler` is invoked during subsequent system event notifications once a subscription has been made with `sysevent_subscribe_event(3SYSEVENT)`.

The system event is represented by the argument `ev` and is passed as an argument to the invoked event delivery function, `event_handler`.

Additional threads are created to service communication between `syseventd(1M)` and the calling process and to run the event handler routine, `event_handler`.

The `sysevent_unbind_handle()` 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 `sysevent_unbind_handle()`.

### RETURN VALUES
The `sysevent_bind_handle()` function returns a valid `sysevent` subscriber handle if the handle is successfully allocated. Otherwise, `NULL` is returned and `errno` is set to indicate the error.

The `sysevent_unbind_handle()` function returns no value.

### ERRORS
The `sysevent_bind_handle()` function will fail if:
- `EACCESS` The calling process has an ID other than the privileged user.
- `EBUSY` There are no resources available.
- `EINVAL` The pointer to the function `event_handler` is `NULL`.
- `EMFILE` The process has too many open descriptors.
- `ENOMEM` There are insufficient resources to allocate the handle.

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

**SEE ALSO**

`syseventd(1M), sysevent_subscribe_event(3SYSEVENT), attributes(5)`
NAME
sysevent_free – free memory for sysevent handle

SYNOPSIS
cc [flag...] file ...-lsysevent [library ...]
#include <libsysevent.h>

void sysevent_free(sysevent_t *ev);

PARAMETERS

ev handle to event an event buffer

DESCRIPTION
The sysevent_free() function deallocates memory associated with an event buffer.

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>

SEE ALSO
attributes(5)
NAME

sysevent_get_attr_list – get attribute list pointer

SYNOPSIS

cc [flag ...] file ... -lsysevent -lnvpair [library ...]
#include <libsysevent.h>
#include <libnvpair.h>

int sysevent_get_attr_list(sysevent_t *ev, nvlist_t **attr_list);

PARAMETERS

ev handle to a system event
attr_list address of a pointer to attribute list (nvlist_t)

DESCRIPTION

The sysevent_get_attr_list() function updates attr_list to point to a searchable name-value pair list associated with the sysevent event, ev. 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 nvlist_free(). See nvlist_alloc(3NVPAIR).

RETURN VALUES

The sysevent_get_attr_list() function returns 0 if the attribute list for ev is found to be valid. Otherwise it returns −1 and sets errno to indicate the error.

ERRORS

The sysevent_get_attr_list() function will fail if:

ENOMEM Insufficient memory available to allocate an nvlist.
EINVAL Invalid sysevent event attribute list.

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>

SEE ALSO

syseventd(1M), nvlist_alloc(3NVPAIR), nvlist_lookup_boolean(3NVPAIR), attributes(5)
### 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
```
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);
```

### PARAMETERS
- `eid` pointer to `sysevent_id_t` structure
- `ev` 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 `ev`.

The `sysevent_get_event_id()` function returns the unique event identifier associated with the `sysevent` handle, `ev`. The identifier is composed of a relative timestamp issued at the time the event was generated and a sequence number to ensure uniqueness.

```
typedef struct sysevent_id {
    uint64_t eid_seq;
    hrtime_t eid_ts;
} sysevent_id_t;
```

The `sysevent_get_size()` function returns the size of the event buffer, `ev`.

### EXAMPLES
#### EXAMPLE 1 parse sysevent header information.

The following example parses sysevent header information from an application’s event handler.

```
hrtimet last_ev_time;
unit64t 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;
    }
```

---

544 man pages section 3: Extended Library Functions • Last Revised 12 Sep 2000
EXAMPLE 1 Parse sysevent header information. (Continued)

```c
/*
 * 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:

<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>

**SEE ALSO** attributes(5)
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)`.

**EXAMPLE 1** Parse sysevent header information.

The following example parses sysevent header information from an application's event handler.

```c
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);
    }
}```
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:

<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>

SEE ALSO malloc(3MALLOC), attributes(5)
NAME: sysevent_post_event – post system event for applications

SYNOPSIS:

cc [flag...] file ... -lsysevent -lnvpair [library...]
#include <libsyevent.h>
#include <libnvpair.h>

int sysevent_post_event(char *class, char *subclass, char *vendor, char *publisher, nvlist_t *attr_list, sysevent_id_t *eid);

PARAMETERS:

attr_list 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

class pointer to a string defining the event class

eid pointer to a system unique identifier

publisher pointer to a string defining the event’s publisher name

subclass pointer to a string defining the event subclass

vendor pointer to a string defining the vendor

DESCRIPTION:

The sysevent_post_event() 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 syseventd(1M).

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 devfsadm(1M) has a publisher string of devfsadm.

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 eid.

RETURN VALUES:

The sysevent_post_event() function returns 0 if the system event has been queued successfully for delivery. Otherwise it returns −1 and sets errno to indicate the error.

ERRORS:

The sysevent_post_event() function will fail if:

ENOMEM Insufficient resources to queue the system event.

EIO The syseventd daemon is not responding and events cannot be queued or delivered at this time.

EINVAL Invalid argument.

EPERM Permission denied.
A copy error occurred.

**EXAMPLE 1**  
Post a system event event with no attributes.

The following example posts a system event event with no attributes.

```c
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.

```c
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:

<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>

**SEE ALSO**  
devfsadm(1M), syseventd(1M), nvlist_add_boolean(3NVPAIR), nvlist_alloc(3NVPAIR), attributes(5)
The `sysevent_subscribe_event()` function registers the caller’s interest in event notifications belonging to the class `event_class` and the subclasses contained in `event_subclass_list`. The subscriber handle `sysevent_hdl` is updated with the new subscription and the calling process receives event notifications from the event handler specified in `sysevent_bind_handle`.

System events matching `event_class` and a subclass contained in `event_subclass_list` published after the caller returns from `sysevent_subscribe_event()` are guaranteed to be delivered to the calling process. Matching system events published and queued prior to a call to `sysevent_subscribe_event()` may be delivered to the process’s event handler.

The `num_subclasses` argument provides the number of subclass string elements in `event_subclass_list`.

A caller can use the event class `SE_ALL_CLASSES` to subscribe to all event classes and subclasses. The event class `SE_ALL_SUBCLASSES` can be used to subscribe to all subclasses within a given event class.

Subsequent calls to `sysevent_subscribe_event()` are allowed to add additional classes or subclasses. To remove an existing subscription, `sysevent_unsubscribe_event()` must be used to remove the subscription.

The `sysevent_unsubscribe_event()` function removes the subscription described by `event_class` for `sysevent_hdl`. Event notifications matching `event_class` will not be delivered to the calling process upon return.

A caller can use the event class `SE_ALL_CLASSES` to remove all subscriptions for `sysevent_hdl`.

The library manages all subscription resources.
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 `sysevent_hdl` argument is an invalid `sysevent` handle.
- **ENOMEM** There is insufficient memory available to allocate subscription resources.

**EXAMPLE 1** Subscribing for DR and environmental events

```c
/* ARGSUSED */
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);
        }
    }
    nvlist_free(nvlist);
}
```

```c
#define MAX_SUBCLASS 3

void
main(int argc, char **argv)
```

---

** 반환값 (RETURN VALUES)**

`sysevent_subscribe_event()` 함수는 구독이 성공적인 경우 0을 반환합니다. 실패할 경우, −1이 반환되고 `errno`는 해당 오류를 나타내는 값을 설정합니다.

**오류 (ERRORS)**

`sysevent_subscribe_event()` 함수가 실패할 경우 다음과 같습니다:
- **EACCESS** 호출 프로세스의 ID가 권한된 사용자와 다른 경우.
- **EINVAL** `sysevent_hdl` 인수는 `sysevent` 핸들을 오래 사용할 경우.
- **ENOMEM** 구독 자원을 할당할 수 없는 메모리부족.

**예시 (EXAMPLES)**

**예시 1** DR 및 환경적 이벤트 구독

```c
/* ARGSUSED */
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);
        }
    }
    nvlist_free(nvlist);
}
```
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[1] = 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:

<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>

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
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double tan(double x);

DESCRIPTION
The tan() function computes the tangent of its argument x, measured in radians.

RETURN VALUES
Upon successful completion, tan() returns the tangent of x.

If x is NaN or ±Inf, NaN is returned.

ERRORS
No errors will occur.

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>

SEE ALSO
atan(3M), isnan(3M), attributes(5)
tanh(3M)

NAME
tanh – hyperbolic tangent function

SYNOPSIS
cc [ flag ... ] file ... -lm [ library ... ]
#include <math.h>

double tanh(double x);

DESCRIPTION
The tanh() function computes the hyperbolic tangent of x.

RETURN VALUES
Upon successful completion, tanh() returns the hyperbolic tangent of x.
If x is NaN, NaN is returned.

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>

SEE ALSO
atanh(3M), isnan(3M), tan(3M), attributes(5)
 NAME tnfctl_buffer_alloc, tnfctl_buffer_dealloc – allocate or deallocate a buffer for trace data

 SYNOPSIS

 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);

 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), trace_file_name 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 trace_file_name. 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, trace_file_name is ignored. Use tnfxtract(1) to extract a kernel
buffer into a file.

trace_buffer_size 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. hndl 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():

 TNFCTL_ERR_BUFEVENTS A buffer already exists.

 TNFCTL_ERR_ACCESS Permission denied; could not create a trace
   file.

 TNFCTL_ERR_SIZETOOSMALL The trace_buffer_size requested is smaller
   than the minimum trace buffer size needed.
Use `trace_min_size` of trace attributes in `tnfctl_trace_attrs_get(3TNF)` to determine the minimum size of the buffer.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNFCTL_ERR_SIZETOOBIG</td>
<td>The requested trace file size is too big.</td>
</tr>
<tr>
<td>TNFCTL_ERR_BADARG</td>
<td><code>trace_file_name</code> is NULL or the absolute path name is longer than MAXPATHLEN.</td>
</tr>
<tr>
<td>TNFCTL_ERR_ALLOCFAIL</td>
<td>A memory allocation failure occurred.</td>
</tr>
<tr>
<td>TNFCTL_ERR_INTERNAL</td>
<td>An internal error occurred.</td>
</tr>
</tbody>
</table>

The following error codes apply to `tnfctl_buffer_dealloc()`:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNFCTL_ERR_BADARG</td>
<td><code>hndl</code> is not a kernel handle.</td>
</tr>
<tr>
<td>TNFCTL_ERR_NOBUF</td>
<td>No buffer exists to deallocate.</td>
</tr>
<tr>
<td>TNFCTL_ERR_BADDEALLOC</td>
<td>Cannot deallocate a trace buffer unless tracing is stopped. Use <code>tnfctl_trace_state_set(3TNF)</code> to stop tracing.</td>
</tr>
<tr>
<td>TNFCTL_ERR_INTERNAL</td>
<td>An internal error occurred.</td>
</tr>
</tbody>
</table>

**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>

**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)`
tnfctl_close – close a tnfctl handle

#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_close(tnfctl_handle_t *hndl, tnfctl_targ_op_t action);

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 action argument is only used in direct mode, that is, if hndl was created by tnfctl_exec_open(3TNF) or tnfctl_pid_open(3TNF). In direct mode, action specifies whether the process will proceed, be killed, or remain suspended. action may have the following values:

- **TNFCTL_TARG_DEFAULT**: Kills the target process if hndl 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():

- **TNFCTL_ERR_BADARG**: A bad argument was sent in action.
- **TNFCTL_ERR_INTERNAL**: An internal 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>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
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)
The interfaces `tnfctl_indirect_open()` and `tnfctl_check_libs()` are used to control probes in another process where the `libtnfctl(3TNF)` client has already opened `/proc(4)` on the target process. An example of this is when the client is a debugger. Since these clients already use `/proc` on the target, `libtnfctl(3TNF)` cannot use `/proc` 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 `libtnfprobe.so.1` (defined in `<tnf/tnfctl.h>` as macro `TNFCTL_LIBTNFPROBE`).

The first argument `prochandle` 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 `/proc` on the target process (the `/proc` file descriptor). The second argument, `config`, is a pointer to

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;

The first field `p_read` is the address of a function that can read `size` bytes at address `addr` in the target image into the buffer `buf`. The function should return 0 upon success. The second field `p_Write` is the address of a function that can write `size` bytes at address `addr` in the target image from the buffer `buf`. The function should return 0 upon success. The third field `p_getpid` is the address of a function that should return the process id of the target process (`prochandle`). The fourth field `p_obj_iter` is the address of a function that iterates over all load objects and the executable by calling the callback function `func` with `client_data`. If `func` returns 0, `p_obj_iter` should continue processing link objects. If `func` returns any other value, `p_obj_iter` should stop calling the callback function and return that value. `p_obj_iter` should return 0 if it iterates over all load objects.

If a failure is returned by any of the functions in `config`, the error is propagated back as `PREX_ERR_INTERNAL` by the `libtnfctl` interface that called it.

The definition of `tnfctl_ind_obj_f` is:
typedef int
tnfctl_ind_obj_f(void *prochandle,
    const struct tnfctl_ind_obj_info *obj
    void *client_data);

typeid 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间接_open() and tnfctl_check_libs() return
TNFCTL_ERR_NONE upon success.

ERRORS

The following error codes apply to tnfctl间接_open():

TNFCTL_ERR_ALLOCFAIL A memory allocation failure occurred.

TNFCTL_ERR_BUSY Internal tracing is being used.

TNFCTL_ERR_NOLIBTNFPROBE libtnfprobe.so.1 is not loaded in the
target process.

TNFCTL_ERR_INTERNAL An internal error occurred.

The following error codes apply to tnfctl_check_libs():

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtncf</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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() 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.
NAME
 tnftctl_internal_open - create handle for internal process probe control

SYNOPSIS
e [ flag . . . ] file . . . -ltftctl [ library . . . ]
# include <tnf/tnfctl.h>

tnfctl_errcode_t tnftctl_internal_open(tnfctl_handle_t **ret_val);

DESCRIPTION
 tnftctl_internal_open() returns in ret_val 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
 tnftctl_internal_open() returns TNFCTL_ERR_NONE upon success.

ERRORS
 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>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO
 ld(1), prex(1), TNF_PROBE(3TNF), dlopen(3DL), dlclose(3DL),
 libtnfctl(3TNF), tracing(3TNF), attributes(5)

Linker and Libraries Guide

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,
 tnftctl_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
ee [ flag ... ] file ... -ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

```c
    tnfctl_errcode_t tnfctl_kernel_open(tnfctl_handle_t **ret_val);
```

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

```
fuser -f /dev/tnfctl
```
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

- **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>

SEE ALSO

prex(1), fuser(1M), TNF_PROBE(3TNF), libtnfctl(3TNF), tracing(3TNF),
tnf_kernel_probes (4), attributes(5)
The `tnfctl_pid_open()` function attaches to a running process with process id of `pid`. The process is stopped on return of this call. The `tnfctl_pid_open()` function returns an error message if `pid` is the same as the calling process. See `tnfctl_internal_open(3TNF)` for information on internal process probe control. A pointer to an opaque handle is returned in `ret_val`, which can be used to control the process and the probes in the process. The target process must have `libtnfprobe.so.1` (defined in `<tnf/tnfctl.h>` as macro `TNFCTL_LIBTNFPROBE`) linked in for probe control to work.

The `tnfctl_exec_open()` function is used to `exec(2)` a program and obtain a probe control handle. For probe control to work, the process image to be `exec`ed must load `libtnfprobe.so.1`. The `tnfctl_exec_open()` function makes it simple for the library to be loaded at process start up time. The `pgm_name` argument is the command to `exec`. If `pgm_name` is not an absolute path, then the `$PATH` environment variable is used to find the `pgm_name`. `argv` 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 `argv` argument must have at least one member, and it should point to a string that is the same as `pgm_name`. See `execve(2)`. The `libnfpbopre_path` argument is an optional argument, and if set, it should be the path to the directory that contains `libtnfprobe.so.1`. There is no need for a trailing "/" in this argument. This argument is useful if `libtnfprobe.so.1` is not installed in `/usr/lib`. `ld_preload` is a space-separated list of libraries to preload into the target program. This string should follow the syntax guidelines of the `LD_PRELOAD` environment variable. See `ld.so.1(1)`. The following illustrates how strings are concatenated to form the `LD_PRELOAD` environment variable in the new process image:
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()`:

---

566 man pages section 3: Extended Library Functions • Last Revised 4 Mar 1997
TNFCTL_ERR_BADARG The pid specified is the same process. Use tnfctl_internal_open(3TNF) 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 /proc 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 libtnfprobe.so.1 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 libtnfprobe.so.1 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. hndl is not a direct process probe control handle.

TNFCTL_ERR_INTERNAL An internal error occurred.

TNFCTL_ERR_NOPROCESS No such target process exists.

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.

```c
const char *pgm;
char * const *argv;
tnfctl_handle_t *hndl, *new_hndl, *child_hndl;
tnfctl_errcode_t err;
```

Extended Library Functions 567
**EXAMPLE 1 Using tnfctl_pid_open()**

```c
char * const *envptr;
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.

```c
/* 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`.

```c
/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, 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.

```c
/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
/* envptr set up to caller's needs */
err = tnfctl_exec_open(pgm, argv, envptr, NULL, NULL, &hndl);
```

Use this example to resume a process that does an `exec(2)` without controlling it.

```c
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)`. 

```c
```
EXAMPLE 1 Using `tnfctl_pid_open()` (Continued)

```c
/*
 * 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()`:

```c
err = tnfctl_continue(hndl, &evt, NULL);
```

The next example is how to control child processes that `fork(2)` or `fork1(2)` create.

```c
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:

<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>

**SEE ALSO**

`ld(1), prex(1), proc(1), exec(2), execve(2), exit(2), fork(2), T NF_PROBE(3TNF), dlclose(3DL), dlopen(3DL), libtnfctl(3TNF), tnfctl_close(3TNF), tnfctl_internal_open(3TNF), tracing(3TNF) attributes(5)`

**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.
**NAME**
tnftcl_probe_apply, tnfctl_probe_apply_ids – iterate over probes

**SYNOPSIS**
```
cc [ flag ... ] file ... -ltnftcl [ 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);
```

**DESCRIPTION**

`tnfctl_probe_apply()` is used to iterate over the probes controlled by `hndl`. For every probe, the `probe_op` function is called:

```c
typedef tnfctl_errcode_t (*tnfctl_probe_op_t)(
    tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hndl,
    void *clientdata);
```

Several predefined functions are available for use as `probe_op`. These functions are described in `tnfctl_probe_state_get(3TNF)`.

The `clientdata` supplied in `tnfctl_probe_apply()` is passed in as the last argument of `probe_op`. The `probe_hndl` in the probe operation function can be used to query or change the state of the probe. See `tnfctl_probe_state_get(3TNF)`. The `probe_op` 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 `probe_hndl` is the same as the lifetime of `hndl`. It is good until `hndl` is closed by `tnfctl_close(3TNF)`. Do not confuse a `probe_hndl` with `hndl`. The `probe_hndl` refers to a particular probe, while `hndl` refers to a process or the kernel. If `probe_hndl` 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 `probe_op` is called only for probes that match a probe id specified in the array of integers referenced by `probe_ids`. The number of probe ids in the array should be specified in `probe_count`. Use `tnfctl_probe_state_get()` to get the `probe_id` that corresponds to the `probe_hndl`.

**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()`:

- `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:

```c
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:

```c
/* 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:

<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>
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

`cc [ flag ... ] file ... -ltncctl [ library ... ]`

```c
#include <tnf/tnfctl.h>

tnfctl_errcode_t tnfctl_probe_state_get(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hdl, tnfctl_probe_state_t *state);

tnfctl_errcode_t tnfctl_probe_enable(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hdl, void *ignored);

tnfctl_errcode_t tnfctl_probe_disable(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hdl, void *ignored);

tnfctl_errcode_t tnfctl_probe_trace(tnfctl_handle_t *hndl,
    tnfctl Probe_t *probe_hdl, void *ignored);

tnfctl_errcode_t tnfctl_probe_untrace(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hdl, void *ignored);

tnfctl_errcode_t tnfctl Probe_disconnect_all(tnfctl_handle_t *hndl,
    tnfctl Probe_t *probe_hdl, void *ignored);

tnfctl_errcode_t tnfctl_probe_connect(tnfctl_handle_t *hndl,
    tnfctl_probe_t *probe_hdl, const char *lib_base_name, const char
    *func_name);
```

DESCRIPTION

`tnfctl_probe_state_get()` returns the state of the probe specified by `probe_hdl` in
the process or kernel specified by `hndl`. The user will pass these in to an apply
iterator. The caller must also allocate `state` and pass in a pointer to it. The semantics of
the individual members of `state` are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique integer assigned to this probe. This number</td>
</tr>
<tr>
<td></td>
<td>does not change over the lifetime of this probe. A</td>
</tr>
<tr>
<td></td>
<td><code>probe_hdl</code> can be obtained by using the calls</td>
</tr>
<tr>
<td></td>
<td><code>tnfctl_apply()</code>, <code>tnfctl_apply_ids()</code>, or</td>
</tr>
<tr>
<td></td>
<td><code>tnfctl_register_funcs()</code>.</td>
</tr>
<tr>
<td>attr_string</td>
<td>A string that consists of <code>attribute value</code> pairs</td>
</tr>
<tr>
<td></td>
<td>separated by semicolons. For the syntax of this string,</td>
</tr>
<tr>
<td></td>
<td>see the syntax of the <code>detail</code> argument of the</td>
</tr>
<tr>
<td></td>
<td><code>TNF_PROBE(3TNF)</code> macro. The attributes <code>name</code>, <code>slots</code>,</td>
</tr>
<tr>
<td></td>
<td><code>keys</code>, <code>file</code>, and <code>line</code> are defined for every probe.</td>
</tr>
<tr>
<td></td>
<td>Additional user-defined attributes can be added by</td>
</tr>
<tr>
<td></td>
<td>using the <code>detail</code> argument of the <code>TNF_PROBE(3TNF)</code></td>
</tr>
<tr>
<td></td>
<td>macro. An example of <code>attr_string</code> follows:</td>
</tr>
<tr>
<td></td>
<td>`*name pageout;slots vnode pages_pageout ;</td>
</tr>
<tr>
<td></td>
<td>keys vm pageio io;file vm.c;line 25;*`</td>
</tr>
<tr>
<td>enabled</td>
<td>B_TRUE if the probe is enabled, or B_FALSE if the</td>
</tr>
<tr>
<td></td>
<td>probe is disabled. Probes are disabled by default. Use</td>
</tr>
</tbody>
</table>
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 probe_op field of **tnfctl Probe**(3TNF) and **tnfctl Probe apply ids**(3TNF). **tnfctl Probe enable( )** enables the probe specified by probe_hndl. 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 probe_hndl.
tnfctl_probe_trace() turns on tracing for the probe specified by probe_hndl. Probes emit a trace record only if the probe is traced.

tnfctl_probe_untrace() turns off tracing for the probe specified by probe_hndl. This is useful if you want to connect probe functions to a probe without tracing it.
	nfctl_probe_connect() connects the function func_name which exists in the library lib_base_name, to the probe specified by probe_hndl.
	nfctl_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.
	nfctl_probe_disconnect_all() disconnects all probe functions from the probe specified by probe_hndl.

Note that no libtnfctl call returns a probe handle (tnfctl_probe_t), yet each of the routines described here takes a probe_hndl 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_hndl argument when using any of the functions described here.

RETURN VALUES

tnfctl_probe_state_get(), tnfctl_probe_enable(),

ERRORS

The following error codes apply to tnfctl_probe_state_get():

TNFCTL_ERR_INVALIDPROBE probe_hndl is no longer valid. The library that the probe was in could have been dynamically closed by dlclose(3DL).

The following error codes apply to tnfctl_probe_enable(),

TNFCTL_ERR_INVALIDPROBE probe_hndl is no longer valid. The library that the probe was in could have been dynamically closed by dlclose(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_hndl 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:

<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>

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)
The function tnfctl_register_funcs() is used to store client-specific data on a per-probe basis. It registers a creator and a destructor function with hndl, either of which can be NULL. The creator function is called for every probe that currently exists in hndl. Every time a new probe is discovered, that is brought in by dlopen(3DL), create_func is called.

The return value of the creator function is stored as part of the probe state and can be retrieved by tnfctl_probe_state_get(3TNF) in the member field client_registered_data.

destroy_func is called for every probe handle that is freed. This does not necessarily happen at the time dlclose(3DL) frees the shared object. The probe handles are freed only when hndl is closed by tnfctl_close(3TNF). If tnfctl_register_funcs() is called a second time for the same hndl, then the previously registered destructor function is called first for all of the probes.

RETURN VALUES

tnfctl_register_funcs() returns T NFCTL_ERR_NONE upon success.

ERRORS

TNFCTL_ERR_INTERNAL An internal 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>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

SEE ALSO

prex(1), T NF_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_strerror(3TNF)

NAME
tnfctl_strerror – map a tnfctl error code to a string

SYNOPSIS
const char * tnfctl_strerror(tnfctl_errcode_t errcode);

DESCRIPTION
tnfctl_strerror() maps the error number in errcode 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>

SEE ALSO prex(1), TNF_PROBE(3TNF), libtnfctl(3TNF), tracing(3TNF), attributes(5)
tnfctl_trace_attrs_get – get the trace attributes from a tnfctl handle

SYNOPSIS

```
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);
```

DESCRIPTION

The `tnfctl_trace_attrs_get()` function returns the trace attributes associated with `hndl` in `attrs`. The trace attributes can be changed by some of the other interfaces in `libtnfctl(3TNF)`. It is the client's responsibility to use `tnfctl_trace_attrs_get()` to get the new trace attributes after use of interfaces that change them. Typically, a client will use `tnfctl_trace_attrs_get()` after a call to `tnfctl_continue(3TNF)` in order to make sure that tracing is still working. See the discussion of `trace_buf_state` that follows.

Trace attributes are represented by the `struct tnfctl_trace_attrs` structure defined in `<tnf/tnfctl.h>`:

```
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;
};
```

The semantics of the individual members of `attrs` are:

- **targ_pid**
  The process id of the target process. This is not valid for kernel tracing.

- **trace_file_name**
  The name of the trace file to which the target writes. `trace_file_name` will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other `libtnfctl` interfaces. The client should copy this string if it should be saved for the use of other `libtnfctl` interfaces.

- **trace_buf_size**
  The size of the trace buffer or file in bytes.

- **trace_min_size**
  The minimum size in bytes of the trace buffer that can be allocated by using the `tnfctl_buffer_alloc(3TNF)` interface.

- **trace_buf_state**
  The state of the trace buffer. `TNFCTL_BUF_OK` indicates that a trace buffer has been allocated. `TNFCTL_BUF_NONE` indicates that no buffer has been allocated. `TNFCTL_BUF_BROKEN` indicates that there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will be written.
tnfctl_trace_attrs_get(3TNF)

be written. To fix tracing, restart the process. For kernel
tracing, deallocate the existing buffer with
tnfctl_buffer_dealloc(3TNF) and allocate a new
one with tnfctl_buffer_alloc(3TNF).

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 tnfctl_trace_state_set(3TNF). For a
process, this state is on by default and can only be
changed by tnf_process_disable(3TNF) and
tnf_process_enable(3TNF).

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
tnfctl_filter_list_get(3TNF),
tnfctl_filter_list_add(3TNF), and
tnfctl_filter_list_delete(3TNF). 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
tnfctl_filter_state_set(3TNF) 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:

TNFCTL_ERR_INTERNAL

An internal 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>Availability</td>
<td>SUNWtnfc</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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)
NAME

The interfaces to control kernel tracing and process filtering are used only with kernel handles, handles created by tnfctl_kernel_open(3TNF). These interfaces are used to change the tracing and filter states for kernel tracing.

tnfctl_trace_state_set() sets the kernel global tracing state to "on" if trace_state is B_TRUE, or to "off" if trace_state is B_FALSE. For the kernel, trace_state is off by default. Probes that are enabled will not write out data unless this state is on. Use tnfctl_trace_attrs_get(3TNF) to retrieve the current tracing state.

tnfctl_filter_state_set() sets the kernel process filtering state to "on" if filter_state is B_TRUE, or to "off" if filter_state is B_FALSE. filter_state is off by default. If it is on, only probe points encountered by processes in the process filter set by tnfctl_filter_list_add() will generate trace points. Use tnfctl_trace_attrs_get(3TNF) to retrieve the current process filtering state.

tnfctl_filter_list_get() returns the process filter list as an array in pid_list. The count of elements in the process filter list is returned in pid_count. The caller should use free(3C) to free memory allocated for the array pid_list.

tnfctl_filter_list_add() adds pid_to_add 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.

tnfctl_filter_list_delete() deletes pid_to_delete 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 tnfctl_trace_state_set(), tnfctl_filter_state_set(), tnfctl_filter_list_add(), tnfctl_filter_list_delete(), and tnfctl_filter_list_get() return TNFCTL_ERR_NONE upon success.

ERRORS

The following error codes apply to tnfctl_trace_state_set:

TNFCTL_ERR_NONE
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_BUF BROKEN      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:

<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>

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, 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

### NAME

TNF_DECLARE_RECORD, TNF_DEFINE_RECORD_1, TNF_DEFINE_RECORD_2, TNF_DEFINE_RECORD_3, TNF_DEFINE_RECORD_4, TNF_DEFINE_RECORD_5

### SYNOPSIS

```c
cc [ flag ... ] file ... [ -ltntfprobe ] [ 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);
```

### 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 `tnf_type` 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 `c_type` argument must be a C struct type. It is the template from which the new `tnf_type` is being created. Not all elements of the C struct need be provided in the TNF type being defined.
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.

**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 it's symbols.

```c
#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",
        "sunwdebug 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:
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 *tnfOpaque*.

**SEE ALSO**

prex(1), tnf_dump(1), TNF_PROBE(3TNF), tnf_process_disable(3TNF), attributes(5)

**NOTES**

Extended Library Functions
This macro interface is used to insert probes into C or C++ code for tracing. See `tnf_probe(3TNF)` 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 .init sections, .fini sections, multi-threaded code, shared objects, and shared objects opened by `dlopen(3DL)`. Use probes to generate trace data for performance analysis or to write debugging output to stderr. Probes are controlled at runtime by `prex(1)`.
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 name of the probe should follow the syntax guidelines for identifiers in ANSI C. The use of name 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  keys 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. keys cannot be a variable. It must be a string constant.

detail  detail 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 detail for one of two reasons. First, use detail to supply an attribute that a user can type into prex(1) to select probes. For example, if a user defines an attribute called color, then prex(1) can select probes based on the value of color. Second, use detail to
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Values that prex matches on</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ%debug</td>
<td>'entering function A'</td>
<td>'entering function A'</td>
</tr>
<tr>
<td>XYZ%exception</td>
<td>'no file'</td>
<td>'no file'</td>
</tr>
<tr>
<td>XYZ%func_entry</td>
<td>/.*/</td>
<td>(regular expression)</td>
</tr>
<tr>
<td>XYZ%color</td>
<td>red blue</td>
<td>red &lt;or&gt; blue</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of probe</td>
</tr>
<tr>
<td>keys</td>
<td>keys of the probe (value is space-separated tokens)</td>
</tr>
<tr>
<td>file</td>
<td>file name of the probe</td>
</tr>
<tr>
<td>line</td>
<td>line number of the probe</td>
</tr>
<tr>
<td>slots</td>
<td>slot names of the probe event (arg_name_n)</td>
</tr>
<tr>
<td>object</td>
<td>the executable or shared object that this probe is in.</td>
</tr>
<tr>
<td>debug</td>
<td>distinguishes debug probes from production probes</td>
</tr>
</tbody>
</table>

arg_type_n

This is the type of the nth argument. The following are predefined TNF types:

<table>
<thead>
<tr>
<th>tnf Type</th>
<th>Associated C type (and semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnf_int</td>
<td>int</td>
</tr>
</tbody>
</table>
TNF_PROBE(3TNF)

<table>
<thead>
<tr>
<th>tnf Type</th>
<th>Associated C type (and semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tnf_uint</td>
<td>unsigned int</td>
</tr>
<tr>
<td>tnf_long</td>
<td>long</td>
</tr>
<tr>
<td>tnf_ulong</td>
<td>unsigned long</td>
</tr>
<tr>
<td>tnf_longlong</td>
<td>long long (if implemented in compilation system)</td>
</tr>
<tr>
<td>tnf_ulonglong</td>
<td>unsigned long long (if implemented in compilation system)</td>
</tr>
<tr>
<td>tnf_float</td>
<td>float</td>
</tr>
<tr>
<td>tnf_double</td>
<td>double</td>
</tr>
<tr>
<td>tnf_string</td>
<td>char *</td>
</tr>
<tr>
<td>tnf_opaque</td>
<td>void *</td>
</tr>
</tbody>
</table>

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 nth 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 in 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfd</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>
If attaching to a running program with `prex(1)` to control the probes, compile the program with `-ltnfprobe` 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.
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 prex(1) utility.

The `tnf_process_disable()` function turns off probing for the process. The default process state is to have probing enabled. The `tnf_process_enable()` function turns on probing for the process.

The `tnf_thread_disable()` function turns off probing for the currently running thread. Threads are "born" or created with this state enabled. The `tnf_thread_enable()` 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 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>SUNWtnfd</td>
</tr>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
<tr>
<td>MT-Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

### See Also

prex(1), tnfdump(1), TNF_DECLARE_RECORD(3TNF), TNF_PROBE(3TNF),
attributes(5)

### Notes

A probe is considered enabled only if:

- `prex(1)` has enabled the probe AND
- the process has probing enabled, which is the default or could be set with `tnf_process_enable()` AND
- the thread that hits the probe has probing enabled, which is every thread’s default or could be set with `tnf_thread_enable()`.
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.
tracing – overview of tnf tracing system

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).

tnfxtract 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).
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.

```c
#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;
    TNP_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++;
    TNP_PROBE_1(work_queue, "work queue",
        "XYZ%work_load heavy",
        tnf_long, queue_length, q_length);
    TNP_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
<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:
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWtnfd</td>
</tr>
<tr>
<td>MT Level</td>
<td>MT-Safe</td>
</tr>
</tbody>
</table>

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
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);

DESCRIPTION
The volmgt_acquire() routine reserves the removable media device specified as dev.
volmgt_acquire() operates in two different modes, depending on whether or not Volume Management is running. See vold(1M).

If Volume Management is running, volmgt_acquire() attempts to reserve the removable media device specified as dev. Specify dev as either a symbolic device name (for example, floppy0) or a physical device pathname (for example, /vol/dsk/unnamed_floppy).

If Volume Management is not running, volmgt_acquire() requires callers to specify a physical device pathname for dev. Specifying dev as a symbolic device name is not acceptable. In this mode, volmgt_acquire() relies entirely on the major and minor numbers of the device to determine whether or not the device is reserved.

If dev is free, volmgt_acquire() updates the internal device reservation database with the caller’s process id (pid) and the specified id string.

If dev is reserved by another process, the reservation attempt fails and volmgt_acquire():

- sets errno to EBUSY
- fills the caller’s id value in the array pointed to by err
- fills in the pid to which the pointer pidp points with the pid of the process which holds the reservation, if the supplied pidp is non-zero

If the override ovr is non-zero, the call overrides the device reservation.

RETURN VALUES
Upon successful completion, volmgt_acquire() returns a non-zero value.

Upon failure, volmgt_acquire() returns 0. If the return value is 0, and errno is set to EBUSY, the address pointed to by err contains the string that was specified as id (when the device was reserved by the process holding the reservation).

ERRORS
The volmgt_acquire() routine fails if one or more of the following are true:

EINVAL One of the specified arguments is invalid or missing.
EBUSY dev is already reserved by another process (and ovr was not set to a non-zero value)
EXAMPLE 1 Using volmgt_acquire()

In the following example, Volume Management is running and the first floppy drive is reserved, accessed and released.

```c
#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().

```c
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 */
            ...
        }
    } else {
        /* handle other errors */
        ...
    }
} 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:

<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>
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`.

**SEE ALSO**

`vold(1M), free(3C), malloc(3C), volmgt_release(3VOLMGT), attributes(5)`
NAME
volmgt_check – have Volume Management check for media

SYNOPSIS
cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

int volmgt_check(char *pathname);

DESCRIPTION
This routine asks Volume Management to check the specified pathname 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 pathname fails, or if any of the following is true:
ENXIO Volume Management is not running.
EINVAL 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:
if (volmgt_check(NULL)) {
    (void) printf("Volume Management found media\n");
}

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>

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.
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.
NAME    volmgt_feature_enabled – check whether specific Volume Management features are enabled

SYNOPSIS    cc [ flag ... ] file ... -l volmgt [ library ... ]

#include <volmgt.h>

int volmgt_feature_enabled(char *feat_str);

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 feat_str 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.

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");
}

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>

SEE ALSO volmgt_acquire(3VOLMGT), volmgt_release(3VOLMGT), attributes(5)
NAME
.volmgt_inuse – check whether or not Volume Management is managing a pathname

SYNOPSIS
cc [ flag ... ] file ... -Ivolmgt [ library ... ]
#include <volmgt.h>
int volmgt_inuse(char *pathname);

DESCRIPTION
volmgt_inuse() checks whether Volume Management is managing the specified pathname.

RETURN VALUES
A non-zero value is returned if Volume Management is managing the specified pathname, otherwise 0 is returned.

ERRORS
This routine can fail, returning 0, if a stat(2) of the supplied pathname or an open(2) of /dev/volctl fails, or if any of the following is true:

- ENXIO Volume Management is not running.
- EINTR An interrupt signal was detected while checking for the supplied pathname for use.

EXAMPLES
EXAMPLE 1 Using volmgt_inuse()

To see if Volume Management is managing the first floppy disk:

if (volmgt_inuse("/dev/rdiskette0") != 0) {
  (void) printf("volmgt is managing diskette 0\n");
} else {
  (void) printf("volmgt is NOT managing diskette 0\n");
}

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>

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 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
cc [flag]... file... -lvolgmt [library]...
#include <volmgt.h>

int volmgt_ownspath(char *path);

PARAMETERS
path A string containing the path.

DESCRIPTION
The volmgt_ownspath() function checks to see if a given path 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 path is owned by
Volume Management. It returns 0 if path 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 path, and then returns the id for the piece of
media.

char *path;
...

if (volmgt_running()) {
    if (volmgt_ownspath(path)) {
        (void) printf("id of %s is %lld\n",
                        path, media_getid(path));
    }
}

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>

SEE ALSO
volmgt_root(3VOLMGT), volmgt_running(3VOLMGT), attributes(5)
NAME
volmgt_release – release removable media device reservation

SYNOPSIS
cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

int volmgt_release(char *dev);

DESCRIPTION
The volmgt_release() routine releases the removable media device reservation specified as dev. See volmgt_acquire(3VOLMGT) for a description of dev.

If dev is reserved by the caller, volmgt_release() 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 errno is set to 0.

RETURN VALUES
Upon successful completion, volmgt_release returns a non-zero value. Upon failure, 0 is returned.

ERRORS
On failure, volmgt_release() returns 0, and sets errno for one of the following conditions:

EINVAL  dev was invalid or missing.
EBUSY    dev was not reserved by the caller.

EXAMPLES
EXAMPLE 1 Using volmgt_release()

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, &errp, NULL)) {
    /* handle error case */
    ...
}
/* floppy acquired - now access it */
if (!volmgt_release("floppy0")) {
    /* handle error case */
    ...
}

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>
<tr>
<td>Interface Stability</td>
<td>Stable</td>
</tr>
</tbody>
</table>

SEE ALSO
vold(1M), volmgt_acquire(3VOLMGT), attributes(5)
volmgt_root(3VOLMGT)

NAME  volmgt_root – return the Volume Management root directory

SYNOPSIS  

cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

const char *volmgt_root (void);

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:

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");
}

FILES  
/vol      default location for the Volume Management root directory

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>

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
cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>

int volmgt_running(void);

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:

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:

if (volmgt_running() != 0) {
    (void) printf("Volume Management is running\n");
} else {
    (void) printf("Volume Management is NOT running\n");
}

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>

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.
NAME
volmgt_symname, volmgt_symdev – convert between Volume Management symbolic
names, and the devices that correspond to them

SYNOPSIS
cc [ flag ... ] file ... -lvolmgt [ library ... ]
#include <volmgt.h>
char *volmgt_symname(char *pathname);
char *volmgt_symdev(char *symname);

DESCRIPTION
These two routines compliment each other, translating between Volume
Management’s symbolic name for a device, called a symname, and the /dev pathname
for that same device.

volmgt_symname() converts a supplied /dev pathname to a symname, 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 symname,
Volume Management’s idea of a device’s symbolic name for a volume, to the /dev
pathname 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 /dev/volctl fails, or if any of the
following is true:

ENXIO Volume Management is not running.
EINTR An interrupt signal was detected while trying to convert the
supplied pathname to a symname.

volmgt_symdev() can fail if an open(2) of /dev/volctl fails, or if any of the
following is true:

ENXIO Volume Management is not running.
EINTR An interrupt signal was detected while trying to convert the
supplied symname to a /dev pathname.

EXAMPLES
EXAMPLE 1 Testing Floppies
The following tests how many floppies Volume Management currently sees in floppy
drives (up to 10):

```c
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",
```

608 man pages section 3: Extended Library Functions • Last Revised 31 Dec 1996
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/rdsk/c0t6d0s2:

```c
if ((nm = volmgt_symname("/dev/rdsk/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:

<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>

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
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);

DESCRIPTION
The wsreg_add_child_component() function adds the component specified by
childComp to the list of child components contained in the component specified by
comp.

The wsreg_remove_child_component() function removes the component
specified by childComp from the list of child components contained in the component
specified by comp.

The wsreg_get_child_components() function returns the list of child
components contained in the component specified by comp.

RETURN VALUES
The wsreg_add_child_component() function returns a non-zero value if the
specified child component was successfully added; otherwise, 0 is returned.

The wsreg_remove_child_component() function returns a non-zero value if the
specified child component was successfully removed; otherwise, 0 is returned.

The wsreg_get_child_components() function returns a null-terminated array of
Wsreg_component pointers that represents the specified component’s list of child
components. If the specified component has no child components, NULL is returned.
The resulting array must be released by the caller through a call to
wsreg_free_component_array(). See wsreg_create_component(3WSREG).

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 prodreg 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.

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>Unsafe</td>
</tr>
</tbody>
</table>

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)
The `wsreg_add_compatible_version()` function adds the version string specified by `version` to the list of backward-compatible versions contained in the component specified by `comp`.

The `wsreg_remove_compatible_version()` function removes the version string specified by `version` from the list of backward-compatible versions contained in the component specified by `comp`.

The `wsreg_get_compatible_versions()` function returns the list of backward-compatible versions contained in the component specified by `comp`.

The `wsreg_add_compatible_version()` function returns a non-zero value if the specified backward-compatible version was successfully added; otherwise, 0 is returned.

The `wsreg_remove_compatible_version()` function returns a non-zero value if the specified backward-compatible version was successfully removed; otherwise, 0 is returned.

The `wsreg_get_compatible_versions()` 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.

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.

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>Unsafe</td>
</tr>
</tbody>
</table>
wreg_add_compatible_version(3WSREG)

SEE ALSO
prodreg(1M), wsreg_initialize(3WSREG), wsreg_register(3WSREG),
wreg_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   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);

DESCRIPTION The wsreg_add_dependent_component() function adds the component specified  
                by dependentComp to the list of dependent components contained in the component  
                specified by comp.

          The wsreg_remove_dependent_component() function removes the component  
                specified by dependentComp from the list of dependent components contained in the 
                component specified by comp.

          The wsreg_get_dependent_components() function returns the list of dependent  
                components contained in the component specified by comp.

RETURN VALUES The wsreg_add_dependent_component() function returns a non-zero value if the  
                   specified dependent component was successfully added; otherwise, 0 is returned.

          The wsreg_remove_dependent_component() function returns a non-zero value if the  
                   specified dependent component was successfully removed; otherwise, 0 is  
                   returned.

          The wsreg_get_dependent_components() function returns a null-terminated  
                   array of Wsreg_component pointers that represents the specified component’s list of  
                   dependent components. If the specified component has no dependent components,  
                   NULL is returned. The resulting array must be released by the caller through a call to  
                   wsreg_free_component_array(). See wsreg_create_component(3WSREG).

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.
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:

<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>

**SEE ALSO**

wsreg_add_required_component(3WSREG),
wsreg_can_access_registry(3WSREG), wsreg_create_component(3WSREG),
wsreg_initialize(3WSREG), wsreg_register(3WSREG), attributes(5)
For each of these functions, the `comp` argument specifies the component on which these functions operate. The `language` argument is the ISO 639 language code identifying a particular display name associated with the specified component.

The `wsreg_add_display_name()` function adds the display name specified by `display_name` to the component specified by `comp`.

The `wsreg_remove_display_name()` function removes a display name from the component specified by `comp`.

The `wsreg_get_display_name()` function returns a display name from the component specified by `comp`.

The `wsreg_get_display_languages()` returns the ISO 639 language codes for which display names are available from the component specified by `comp`.

The `wsreg_add_display_name()` function returns a non-zero value if the display name was set correctly; otherwise 0 is returned.

The `wsreg_remove_display_name()` function returns a non-zero value if the display name was removed; otherwise 0 is returned.

The `wsreg_get_display_name()` 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 `wsreg_get_display_languages()` 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.

See `attributes(5)` for descriptions of the following attributes:
wsreg_add_display_name(3WSREG)

<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>

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
c {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);

DESCRIPTION
The wsreg_add_required_component() function adds the component specified by
requiredComp to the list of required components contained in the component
specified by comp.

The wsreg_remove_required_component() function removes the component
specified by requiredComp from the list of required components contained in the
component specified by comp.

The wsreg_get_required_components() function returns the list of required
components contained in the component specified by comp.

RETURN VALUES
The wsreg_add_required_component() function returns a non-zero value if the
specified required component was successfully added. Otherwise, 0 is returned.

The wsreg_remove_required_component() function returns a non-zero value if
the specified required component was successfully removed. Otherwise, 0 is returned.

The wsreg_get_required_components() function returns a null-terminated
array of Wsreg_component pointers that represents the specified component’s list of
required components. If the specified component has no required components, NULL
is returned. The resulting array must be released by the caller through a call to
wsreg_free_component_array(). See wsreg_create_component(3WSREG).

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.
ATTTIBUTES

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

wsreg_add_dependent_component(3WSREG),
wsreg_can_access_registry(3WSREG),
wsreg_create_component(3WSREG),
wsreg_initialize(3WSREG),
wsreg_register(3WSREG),
attributes(5)
The `wsreg_can_access_registry()` function is used to determine what access, if any, an application has to the product install registry.

The `access_flag` argument can be one of the following:

- `O_RDONLY` Inquire about read only access to the registry.
- `O_RDWR` Inquire about modify (read and write) access to the registry.

The `wsreg_can_access_registry()` function returns non-zero if the specified access level is permitted. A return value of 0 indicates the specified access level is not permitted.

**EXAMPLE 1** Initialize the registry and determine if access to the registry is permitted.

```c
#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");
    }
}
```

The `wsreg_initialize(3WSREG)` function must be called before calls to `wsreg_can_access_registry()` can be made.

**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>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  
wsreg_initialize(3WSREG), attributes(5)
**wsreg_clone_component(3WSREG)**

**NAME**  
wsreg_clone_component – clone a component

**SYNOPSIS**  
cc [flag ...] file ...-lwsreg [library ...]  
#include <wsreg.h>

Wsreg_component *wsreg_clone_component(const Wsreg_component *comp);

**DESCRIPTION**  
The `wsreg_clone_component()` function clones the component specified by `comp`.

**RETURN VALUES**  
The `wsreg_clone_component()` returns a pointer to a component that is configured exactly the same as the component specified by `comp`.

**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>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-Level</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**  
`wsreg_create_component(3WSREG)`, `wsreg_initialize(3WSREG)`, `wsreg_get(3WSREG)`, attributes(5)
The `wsreg_components_equal()` function determines if the component specified by the `comp1` argument is equal to the component specified by the `comp2` argument. Equality is evaluated based only on the content of the two components, not the order in which data was set into the components.

The `wsreg_components_equal()` function returns a non-zero value if the component specified by the `comp1` argument is equal to the component specified by the `comp2` argument. Otherwise, 0 is returned.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO `wsreg_clone_component(3WSREG), wsreg_create_component(3WSREG), wsreg_initialize(3WSREG), attributes(5)`
wsreg_create_component(3WSREG)

NAME
wsreg_create_component, wsreg_free_component, wsreg_free_component_array –
create or release a component

SYNOPSIS
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);

DESCRIPTION
The wsreg_create_component() function allocates a new component and assigns
the uuid (universal unique identifier) specified by uuid to the resulting component.

The wsreg_free_component() function releases the memory associated with the
component specified by comp.

The wsreg_free_component_array() function frees the null-terminated array of
component pointers specified by complist. This function can be used to free the results
of a call to wsreg_get_all(). See wsreg_get(3WSREG).

RETURN VALUES
The wsreg_create_component() function returns a pointer to the newly allocated
Wsreg_component structure.

The wsreg_free_component_array() function returns a non-zero value if the
specified Wsreg_component array was freed successfully. Otherwise, 0 is returned.

USAGE
A minimal registerable Wsreg_component configuration must include a version,
unique name, display name, and an install location.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_add_display_name(3WSREG), wsreg_get(3WSREG),
wsreg_initialize(3WSREG), wsreg_register(3WSREG),
wsreg_set_id(3WSREG), wsreg_set_location(3WSREG),
wsreg_set_unique_name(3WSREG), wsreg_set_version(3WSREG),
attributes(5)
The \texttt{wsreg_get()} function queries the product install registry for a component that matches the query specified by \texttt{query}.

The \texttt{wsreg_get_all()} function returns all components currently registered in the product install registry.

The \texttt{wsreg_get()} function returns a pointer to a \texttt{Wsreg_component} structure representing the registered component. If no component matching the specified query is currently registered, \texttt{wsreg_get()} returns NULL.

The \texttt{wsreg_get_all()} function returns a null-terminated array of \texttt{Wsreg_component} pointers. Each element in the resulting array represents one registered component.

The \texttt{Wsreg_component} pointer returned from \texttt{wsreg_get()} should be released through a call to \texttt{wsreg_free_component()}. See \texttt{wsreg_create_component(3WSREG)}.

The \texttt{Wsreg_component} pointer array returned from \texttt{wsreg_get_all()} should be released through a call to \texttt{wsreg_free_component_array()}. See \texttt{wsreg_create_component(3WSREG)}.

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

\begin{tabular}{|c|c|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
MT-Level & Unsafe \\
\hline
\end{tabular}

\textbf{SEE ALSO} \texttt{wsreg_create_component(3WSREG)}, \texttt{wsreg_initialize(3WSREG)}, \texttt{wsreg_register(3WSREG)}, attributes\((5)\)
**NAME**
wsreg_initialize – initialize wsreg library

**SYNOPSIS**
cc [flag...] file ...-lwreg [library...]
#include <wsreg.h>

```c
int wsreg_initialize(Wsreg_init_level level, const char *
*alternate_root);
```

**DESCRIPTION**
The `wsreg_initialize()` function initializes the wsreg library.
The `level` argument can be one of the following:

- **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.

The `alternate_root` argument can be used to specify a root prefix. If NULL is specified, no root prefix is used.

**RETURN VALUES**
The `wsreg_initialize()` function can return one of the following:

- **WSREG_SUCCESS**: The initialization was successful and no registry conversion is necessary.
- **WSREG_CONVERSION_RECOMMENDED**: An old registry file exists and should be converted.

A conversion is attempted if the `init_level` argument is **WSREG_INIT_NORMAL** and a registry file from a previous version of the product install registry exists. If the `wsreg_initialize()` 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.

**USAGE**
The `wsreg_initialize()` 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 `/usr/bin/prodreg` or by the registry converter `/usr/bin/regconvert`.

**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>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
`prodreg(1M)`, `wsreg_can_access_registry(3WSREG)`, attributes(5)
wsreg_query_create(3WSREG)

NAME
wsreg_query_create, wsreg_query_free – create a query

SYNOPSIS
cc [flag ...] file ...-lwsreg [library ...]
#include <wsreg.h>
Wsreg_query *wsreg_query_create(void);
void wsreg_query_free(Wsreg_query *query);

DESCRIPTION
The wsreg_query_create() function allocates a new query that can retrieve components from the product install registry.

The wsreg_query_free() function releases the memory associated with the query specified by query.

RETURN VALUES
The wsreg_query_create() 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 wsreg_get(3WSREG) function to perform the registry query.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_get(3WSREG), wsreg_initialize(3WSREG),
wsreg_query_set_id(3WSREG), wsreg_query_set_instance(3WSREG),
wsreg_query_set_location(3WSREG),
wsreg_query_set_unique_name(3WSREG),
wsreg_query_set_version(3WSREG), wsreg_unregister(3WSREG),
attributes(5)
wsreg_query_set_id(3WSREG)

NAME  wsreg_query_set_id, wsreg_query_get_id – set or get the uuid of a query

SYNOPSIS  
cc [flag ...] file ...-lwreg [library ...]
#include <wsreg.h>

int wsreg_query_set_id(Wsreg_query *query, const char *uuid);
char *wsreg_query_get_id(const Wsreg_query *query);

DESCRIPTION  The wsreg_query_set_id() function sets the uuid (universal unique identifier) specified by uuid in the query specified by query. If a uuid has already been set in the specified query, the resources associated with the previously set uuid are released.

The wsreg_query_get_id() function returns the uuid associated with the query specified by query. The resulting string is not a copy and must not be released by the caller.

RETURN VALUES  The wsreg_query_set_id() function returns non-zero if the uuid was set correctly; otherwise 0 is returned.

The wsreg_query_get_id() 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 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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  wsreg_get(3WSREG), wsreg_initialize(3WSREG),
towsreg_query_create(3WSREG), attributes(5)
wsreg_query_set_instance(3WSREG)

NAME        wsreg_query_set_instance, wsreg_query_get_instance – set or get the instance of a query

SYNOPSIS   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);

DESCRIPTION The wsreg_query_set_instance() function sets the instance number specified by
instance in the query specified by query.

The wsreg_query_get_instance() function retrieves the instance from the query
specified by query.

RETURN VALUES The wsreg_query_set_instance() function returns a non-zero value if the
instance was set correctly; otherwise 0 is returned.

The wsreg_query_get_instance() 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 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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO       wsreg_get(3WSREG), wsreg_initialize(3WSREG),
wsreg_query_create(3WSREG), attributes(5)
wsreg_query_set_location(3WSREG)

NAME
wsreg_query_set_location, wsreg_query_get_location – set or get the location of a query

SYNOPSIS
cc [flag ...] file ...-lwreg [library ...]
#include <wreg.h>

int wsreg_query_set_location(Wsreg_query *query, const char *
*location);

cchar *wsreg_query_get_location(Wsreg_query *query);

DESCRIPTION
The wsreg_query_set_location() function sets the location specified by location
in the query specified by query. If a location has already been set in the specified query,
the resources associated with the previously set location are released.

The wsreg_query_get_location() function gets the location string from the
query specified by query.

RETURN VALUES
The wsreg_query_set_location() function returns a non-zero value if the
location was set correctly; otherwise 0 is returned.

The wsreg_query_get_location() 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 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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_get(3WSREG), wsreg_initialize(3WSREG),
wsreg_query_create(3WSREG), attributes(5)
wsreg_query_set_unique_name(3WSREG)

NAME
wsreg_query_set_unique_name, wsreg_query_get_unique_name – set or get the unique name of a query

SYNOPSIS
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);

DESCRIPTION
The wsreg_query_set_unique_name() function sets the unique name specified by unique_name in the query specified by query. If a unique name has already been set in the specified query, the resources associated with the previously set unique name are released.

The wsreg_query_get_unique_name() function gets the unique name string from the query specified by query. The resulting string is not a copy and must not be released by the caller.

RETURN VALUES
The wsreg_query_set_unique_name() function returns a non-zero value if the unique_name was set correctly; otherwise 0 is returned.

The wsreg_query_get_unique_name() function returns a copy of the unique_name 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 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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_get(3WSREG), wsreg_initialize(3WSREG), wsreg_query_create(3WSREG), attributes(5)
wsreg_query_set_version(3WSREG)

NAME  wsreg_query_set_version, wsreg_query_get_version – set or get the version of a query

SYNOPSIS  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);

DESCRIPTION  The wsreg_query_set_version() function sets the version specified by version in the query specified by query. If a version has already been set in the specified query, the resources associated with the previously set version are released.

The wsreg_query_get_version() function gets the version string from the query specified by query. The resulting string is not a copy and must not be released by the caller.

RETURN VALUES  The wsreg_query_set_version() function returns a non-zero value if the version was set correctly; otherwise 0 is returned.

The wsreg_query_get_version() function returns the version from the specified query. If no version has been set, NULL 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 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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  wsreg_get(3WSREG), wsreg_initialize(3WSREG), wsreg_query_create(3WSREG), attributes(5)
wsreg_register(3WSREG)

NAME
wsreg_register – register a component in the product install registry

SYNOPSIS
cc [flag ...] file ... -lwsreg [library ...]
#include <wsreg.h>

int wsreg_register(Wsreg_component *comp);

DESCRIPTION
The wsreg_register() function updates a component in the product install registry.

If comp is already in the product install registry, the call to wsreg_register() results in the currently registered component being updated. Otherwise, comp 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 comp has required components, each required component is updated to reflect the required component relationship.

If comp 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.

#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;
}
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:

<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>

**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

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);

DESCRIPTION

The wsreg_set_data() function adds the key-value pair specified by key and value to the component specified by comp. If value is NULL, the key and current value is removed from the specified component.

The wsreg_get_data() function retrieves the value associated with the key specified by key from the component specified by comp.

The wsreg_get_data_pairs() function returns the list of key-value pairs from the component specified by comp.

RETURN VALUES

The wsreg_set_data() function returns a non-zero value if the specified key-value pair was successfully added. It returns 0 if the addition failed. If NULL is passed as the value, the current key-value pair are removed from the specified component.

The wsreg_get_data() function returns the value associated with the specified key. It returns NULL 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 wsreg_get_data_pairs() 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, NULL 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 wsreg_register(3WSREG) for the modifications to be persistent.

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>Unsafe</td>
</tr>
</tbody>
</table>
SEE ALSO

prodreg(1M), wsreg_initialize(3WSREG), wsreg_register(3WSREG), attributes(5)
wsreg_set_id(3WSREG)

NAME  wsreg_set_id, wsreg_get_id - set or get the uuid of a component

SYNOPSIS  
cc [flag ...] file ... -Iwsreg [library ...]
#include <wsreg.h>

int wsreg_set_id(Wsreg_component *comp, const char *uuid);
char *wsreg_get_id(const Wsreg_component *comp);

DESCRIPTION  The wsreg_set_id() function sets the uuid (universal unique identifier) specified by uuid into the component specified by comp. If a uuid has already been set into the specified component, the resources associated with the previously set uuid are released.

The wsreg_get_id() function returns a copy of the uuid of the component specified by comp. The resulting string must be released by the caller.

RETURN VALUES  The wsreg_set_id() function returns non-zero if the uuid was set correctly; otherwise 0 is returned.

The wsreg_get_id() function returns a copy of the specified component's uuid.

USAGE  Generally, the uuid will be set into a component by the wsreg_create_component(3WSREG) function, so a call to the wsreg_set_id() is not necessary.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  wsreg_create_component(3WSREG), wsreg_initialize(3WSREG), attributes(5)attributes(5)
NAME
wsreg_set_instance, wsreg_get_instance – set or get the instance of a component

SYNOPSIS
cc [flag ...] file ...-lwsreg [library ...]
#include <wsreg.h>

int wsreg_set_instance(Wsreg_component *comp, int instance);
int wsreg_get_instance(Wsreg_component *comp);

DESCRIPTION
The wsreg_set_instance() function sets the instance number specified by instance
of the component specified by comp. The instance number and uuid are used to
uniquely identify any component in the product install registry.

The wsreg_get_instance() function determines the instance number associated
with the component specified by comp.

RETURN VALUES
The wsreg_set_instance() function returns a non-zero value if the instance was
set correctly; otherwise 0 is returned.

The wsreg_get_instance() 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 to get the instance value of a registered
component.

#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));
}
EXAMPLE 1 Get the instance value of a registered component.  

(Continued)

```c
    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:

<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>

**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**
cc [flag...] file ...-lwsreg [library...]
#include <wsreg.h>

```c
int wsreg_set_location(Wsreg_component *comp, const char *location);
char *wsreg_get_location(const Wsreg_component *comp);
```

**DESCRIPTION**
The `wsreg_set_location()` function sets the location specified by `location` into the component specified by `comp`. 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 `wsreg_get_location()` function gets the location string from the component specified by `comp`. The resulting string must be released by the caller.

**RETURN VALUES**
The `wsreg_set_location()` function returns a non-zero value if the location was set correctly; otherwise 0 is returned.

The `wsreg_get_location()` function returns a copy of the location from the specified component.

**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>Unsafe</td>
</tr>
</tbody>
</table>

**SEE ALSO**
wsreg_initialize(3WSREG), attributes(5)
wsreg_set_parent(3WSREG)

NAME
wsreg_set_parent, wsreg_get_parent – set or get the parent of a component

SYNOPSIS
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);

DESCRIPTION
The wsreg_set_parent() function sets the parent specified by parent of the
component specified by comp.

The wsreg_get_parent() function gets the parent of the component specified by
comp.

RETURN VALUES
The wsreg_get_parent() function returns a pointer to a Wsreg_component
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
Wsreg_component pointer with a call to wsreg_free_component(). See
wsreg_create_component(3WSREG).

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 wsreg_set_parent() can be made,
since the parent’s instance number must be known at the time the
wsreg_set_parent() function is called.

A process needing to call wsreg_set_parent() or wsreg_get_parent() must
have read access to the product install registry.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_can_access_registry(3WSREG), wsreg_create_component(3WSREG),
wsreg_initialize(3WSREG), wsreg_register(3WSREG),
wsreg_set_instance(3WSREG), attributes(5)
wsreg_set_type(3WSREG)

NAME     wsreg_set_type, wsreg_get_type — set or get the type of a component

SYNOPSIS  cc [flag...] file ... -lwsreg [library...]
#include <wsreg.h>

    int wsreg_set_type(Wsreg_component *comp, Wsreg_component_type type);

    Wsreg_component_type wsreg_get_type(const Wsreg_component *comp);

DESCRIPTION The wsreg_set_type() function sets the type specified by type in the component specified by comp.

The wsreg_get_type() function retrieves the type from the component specified by comp.

RETURN VALUES The wsreg_set_type() function returns a non-zero value if the type is set successfully; otherwise 0 is returned.

The wsreg_get_type() function returns the type currently set in the component specified by comp.

USAGE The component type is used to indicate whether a Wsreg_component structure represents a product, feature, or component. The type argument can be one of the following:

    WSREG_PRODUCT Indicates the Wsreg_component represents a product.
                        A product is a collection of features and/or components.

    WSREG_FEATURE Indicates the Wsreg_component represents a feature.
                        A feature is a collection of components.

    WSREG_COMPONENT Indicates the Wsreg_component represents a component. A component is a collection of files that may be installed.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO    wsreg_create_component(3WSREG), wsreg_initialize(3WSREG),
             wsreg_register(3WSREG), wsreg_set_instance(3WSREG), attributes(5)
NAME
wsreg_set_uninstaller, wsreg_get_uninstaller – set or get the uninstaller of a
component

SYNOPSIS
cc [flag ...] file ...-1wsreg [library ...]
#include <wsreg.h>

int wsreg_set_uninstaller(Wsreg_component *comp, const char *
uninstaller);

char *wsreg_set_uninstaller(const Wsreg_component *comp);

DESCRIPTION
The wsreg_set_uninstaller() function sets the uninstaller specified by
uninstaller in the component specified by comp. If an uninstaller has already been set in
the specified component, the resources associated with the previously set uninstaller
are released.

The wsreg_get_uninstaller() function gets the uninstaller string from the
component specified by comp. The resulting string must be released by the caller.

RETURN VALUES
The wsreg_set_uninstaller() function returns a non-zero value if the uninstaller
was set correctly; otherwise 0 is returned.

The wsreg_get_uninstaller() 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 prodreg(1M) registry viewer
will provide an uninstall button that will invoke the uninstaller.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
prodreg(1M), wsreg_initialize(3WSREG), attributes(5)
wsreg_set_unique_name(3WSREG)

NAME
wsreg_set_unique_name, wsreg_get_unique_name – set or get the unique name of a component

SYNOPSIS
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);

DESCRIPTION
The wsreg_set_unique_name() function sets the unique name specified by unique_name in the component specified by comp. 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 wsreg_get_unique_name() function gets the unique name string from the component specified by comp. The resulting string must be released by the caller.

RETURN VALUES
The wsreg_set_unique_name() function returns a non-zero value if the unique name was set correctly; otherwise it returns 0.

The wsreg_get_unique_name() function returns a copy of the unique name from the specified component.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO
wsreg_initialize(3WSREG), attributes(5)
NAME | wsreg_set_vendor, wsreg_get_vendor – set or get the vendor of a component

SYNOPSIS

#include <wsreg.h>

int wsreg_set_vendor(Wsreg_component *comp, const char *vendor);
char *wsreg_get_vendor(const Wsreg_component *comp);

DESCRIPTION

The `wsreg_set_vendor()` function sets the vendor specified by `vendor` in the component specified by `comp`. The `vendor` 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 `wsreg_get_vendor()` function gets the vendor string from the component specified by `comp`. The resulting string must be released by the caller.

RETURN VALUES

The `wsreg_set_vendor()` function returns a non-zero value if the vendor was set correctly; otherwise it returns 0.

The `wsreg_get_vendor()` function returns a copy of the vendor from the specified component.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO

wsreg_initialize(3WSREG), attributes(5)
wsreg_set_version(3WSREG)

NAME  wsreg_set_version, wsreg_get_version – set or get the version of a component

SYNOPSIS  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);

DESCRIPTION  The wsreg_set_version() function sets the version specified by version in the
component specified by comp. The version 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 wsreg_get_version() function gets the version string from the component
specified by comp. The resulting string must be released by the caller.

RETURN VALUES  The wsreg_set_version() function returns a non-zero value if the version was set
correctly; otherwise it returns 0.

The wsreg_get_version() function returns a copy of the version from the
specified component.

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>Unsafe</td>
</tr>
</tbody>
</table>

SEE ALSO  wsreg_initialize(3WSREG), attributes(5)
The `wsreg_unregister()` function removes the component specified by `comp` from the product install registry. The component will only be removed if the `comp` argument has a matching uuid, instance, and version.

Usually, the component retrieved through a call to `wsreg_get()` before being passed to the `wsreg_unregister()` 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.

Upon successful completion, a non-zero return value is returned. If the component could not be unregistered, 0 is returned.

**EXAMPLE 1**  Unregister a component.

The following example demonstrates how to unregister a component.

```c
#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);
        }
        /* The component has been removed */
        wsreg_unregister(query);
    }
    return 0;
}
```
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);
}

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:

<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>

SEE ALSO `wsreg_add_dependent_component(3WSREG), wsreg_get(3WSREG),
wsreg_initialize(3WSREG), wsreg_register(3WSREG), attributes(5)`
y0, y1, yn – Bessel functions of the second kind

SYNOPSIS

cc [ flag ... ] file ... -lm [ library ... ]

double y0(double x);
double y1(double x);
double yn(int n, double x);

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:

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>
<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>

SEE ALSO

isnan(3M), j0(3M), matherr(3M), attributes(5), standards(5)
y0(3M)
Index

A
absolute value function — fabs, 201
access a table property —
   picl_get_next_by_col, 399
access a table property —
   picl_get_next_by_row, 399
access CPU performance counters in other
   processes — cpc_pctx_bind_event, 77
access CPU performance counters in other
   processes — cpc_pctx_invalidate, 77
access CPU performance counters in other
   processes — cpc_pctx_rele, 77
access PROM device information —
   di_prom_prop_data, 103
access PROM device information —
   di_prom_prop_name, 103
access PROM device information —
   di_prom_prop_next, 103
access property values and attributes —
   di_prop_bytes, 107
access property values and attributes —
   di_prop_dev, 107
access property values and attributes —
   di_prop_int64, 107
access property values and attributes —
   di_prop_ints, 107
access property values and attributes —
   di_prop_name, 107
access property values and attributes —
   di_prop_strings, 107
access property values and attributes —
   di_prop_type, 107
access project files from Perl — project, 449
aclcheck — check the validity of an ACL, 20
aclfrommode — convert an ACL to or from
   permission bits, 23
aclfromtext — convert internal representation to
   or from external representation, 24
aclsort — sort an ACL, 22
aclto mode — convert an ACL to or from
   permission bits, 23
acltotext — convert internal representation to or
   from external representation, 24
acos — arc cosine function, 26
acosh — inverse hyperbolic functions, 27
add or delete node to or from tree —
   ptree_add_node, 453
add or delete node to or from tree —
   ptree_delete_node, 453
add or remove a backward compatible version
   — wsreg_add_compatible_version, 612
add or remove a backward compatible version
   — wsreg_get_compatible_versions, 612
add or remove a backward compatible version
   — wsreg_remove_compatible_version, 612
add or remove a child component —
   wsreg_add_child_component, 610
add or remove a child component —
   wsreg_get_child_components, 610
add or remove a child component —
   wsreg_remove_child_component, 610
add or remove a dependent component —
   wsreg_add_dependent_component, 614
add or remove a dependent component —
  wsreg_get_dependent_components, 614
add or remove a dependent component —
  wsreg_remove_dependent_component, 614
add or remove a required component —
  wsreg_add_required_component, 618
add or remove a required component —
  wsreg_get_required_components, 618
add or remove a required component —
  wsreg_remove_required_component, 618
add or retrieve a key-value pair —
  wsreg_get_data_pairs, 635
add or retrieve a key-value pair —
  wsreg_get_data, 635
add or retrieve a key-value pair —
  wsreg_set_data, 635
add, remove, or return a localized display name —
  wsreg_add_display_name, 616
add, remove, or return a localized display name —
  wsreg_get_display_languages, 616
add, remove, or return a localized display name —
  wsreg_remove_required_component, 618
add or retrieve a key-value pair —
  wsreg_get_data_pairs, 635
add or retrieve a key-value pair —
  wsreg_get_data, 635
add or retrieve a key-value pair —
  wsreg_set_data, 635
add, remove, or return a localized display name —
  wsreg_add_display_name, 616
add, remove, or return a localized display name —
  wsreg_get_display_languages, 616
add, remove, or return a localized display name —
  wsreg_remove_required_component, 618
advance — regular expression compile and match routines, 479
allocate or deallocate a buffer for trace data —
  tnfctl_buffer_alloc, 555
  tnfctl_buffer_dealloc, 555
allow or disallow a memory segment to be imported by other nodes —
  rsm_memseg_export_publish, 497
allow or disallow a memory segment to be imported by other nodes —
  rsm_memseg_export_republish, 497
allow or disallow a memory segment to be imported by other nodes —
  rsm_memseg_export_unpublish, 497
annotate source code with info for tools —
  NOTE, 328
  _NOTE, 328
arc cosine function — acos, 26
arc sine function — asin, 28
arc tangent function — atan2, 29
arc tangent function — atan, 30
asin — arc sine function, 28
asinh — inverse hyperbolic functions, 27
associate callbacks with process events —
  pctx_set_events, 391
atan — arc tangent function, 30
atan2 — arc tangent function, 29
atanh — inverse hyperbolic functions, 27
au_close — construct audit records, 31
au_open — construct audit records, 31
au_preselect — preselect an audit record, 33
au_to — create audit record tokens, 35
au_to_arg — create audit record tokens, 35
au_to_arg32 — create audit record tokens, 35
au_to_arg64 — create audit record tokens, 35
au_to_attr — create audit record tokens, 35
au_to_cmd — create audit record tokens, 35
au_to_data — create audit record tokens, 35
au_to_groups — create audit record tokens, 35
au_to_in_addr — create audit record tokens, 35
au_to_ipc — create audit record tokens, 35
au_to_ipc_perm — create audit record tokens, 36
au_to_iport — create audit record tokens, 35
au_to_me — create audit record tokens, 35
au_to_new_in_addr — create audit record tokens, 36
au_to_new_process — create audit record tokens, 36
au_to_new_socket — create audit record tokens, 36
au_to_new_subject — create audit record tokens, 36
au_to_newgroups — create audit record tokens, 35
au_to_opaque — create audit record tokens, 35
au_to_path — create audit record tokens, 35
au_to_process — create audit record tokens, 35
au_to_process_ex — create audit record tokens, 35
au_to_return — create audit record tokens, 35
au_to_return32 — create audit record tokens, 35
au_to_return64 — create audit record tokens, 35
au_to_socket — create audit record tokens, 35
au_to_subject — create audit record tokens, 35
au_to_subject_ex — create audit record tokens, 35
au_to_text — create audit record tokens, 35
au_user_mask — get user’s binary preselection mask, 39
au_write — write audit records, 31
audit control file information
— endac, 210
— getacdir, 210
— getacfig, 210
— getacinfo, 210
— getacmin, 210
— getacna, 210
— setac, 210
audit record tokens, manipulating
— au_close, 31
— au_open, 31
— au_preselect, 33
— au_write, 31
authentication information routines for PAM
— pam_get_item, 366
— pam_set_item, 366
authentication transaction routines for PAM
— pam_end, 383
— pam_start, 383

B
base 10 logarithm function — log10, 284
Basic Security Module functions
— au_close, 31
— au_open, 31
— au_preselect, 33
— au_user_mask, 39
— au_write, 31
Bessel functions of the first kind
— j0, 243
— j1, 243
— jn, 243
Bessel functions of the second kind
— y0, 649
— y1, 649
— yn, 649
bgets — read stream up to next delimiter, 41
bind or unbind subscriber handle —
sysevent_bind_handle, 540
bind or unbind subscriber handle —
sysevent_unbind_handle, 540
buffer
split into fields — bufsplit, 43

C
cbrt — cube root function, 44
ceil — ceiling value function, 45
ceiling value function — ceil, 45
change or add a value to the PAM environment
— pam_putenv, 360
check the validity of an ACL — aclcheck, 20
check whether or not Volume Management is
managing a pathname — volmgmt_inuse, 603
check whether specific Volume Management
features are enabled —
volmgmt_feature_enabled, 602
chkauthattr — verify user authorization, 217
class-dependent data translation
— elf32_xlatetof, 156
— elf32_xlatetom, 156
— elf64_xlatetof, 156
— elf64_xlatetom, 156
close a tnfctl handle — tnfctl_close, 557
c ode a component —
wsreg_clone_component, 622
commands
open, close to and from a command —
p2open, p2close, 344
compile — regular expression compile and
match routines, 479
compute natural logarithm — log1p, 285
computes exponential functions — expm1, 200
cfg_admin — configuration administration
interface, 47
cfg_ap_id_cmp — configuration
administration interface, 47
cfg_change_state — configuration
administration interface, 47
cfg_list — configuration administration
interface, 47
cfg_list_ext — configuration administration
interface, 47
cfg_private_func — configuration
administration interface, 47
cfg_stat — configuration administration
interface, 47

Index 653
config_strerror — configuration administration interface, 47
config_test — configuration administration interface, 47
config_unload_libs — configuration administration interface, 47
configuration administration interface —
config_admin, 47
configuration administration interface —
config_ap_id_cmp, 47
configuration administration interface —
config_change_state, 47
configuration administration interface —
config_list_ext, 47
configuration administration interface —
config_list, 47
configuration administration interface —
config_private_func, 47
configuration administration interface —
config_stat, 47
configuration administration interface —
config_strerror, 47
configuration administration interface —
config_test, 47
configuration administration interface —
config_unload_libs, 47
connect to a DMI service provider
— ConnectToServer, 54, 113
construct, read, and write extended accounting records — ea_copy_object, 140
construct, read, and write extended accounting records — ea_copy_object_tree, 140
construct, read, and write extended accounting records — ea_get_creator, 140
construct, read, and write extended accounting records — ea_get_hostname, 140
construct, read, and write extended accounting records — ea_get_object, 140
construct, read, and write extended accounting records — ea_get_object_tree, 140
construct, read, and write extended accounting records — ea_next_object, 140
construct, read, and write extended accounting records — ea_pack_object, 140
construct, read, and write extended accounting records — ea_previous_object, 140
construct, read, and write extended accounting records — ea_unpack_object, 140
connect to a DMI service provider
— ConnectToServer, 54, 113
control kernel tracing and process filtering
— tnfctl_filter_list_add, 581
— tnfctl_filter_list_delete, 581
— tnfctl_filter_list_get, 581
— tnfctl_filter_state_set, 581
— tnfctl_trace_state_set, 581
control probes of another process where caller provides /proc functionality
— tnfctl_check_libs, 559
— tnfctl_indirect_open, 559
convert an ACL to or from permission bits — aclfrommode, 23
convert an ACL to or from permission bits — acltomode, 23
convert internal representation to or from external representation — aclfromtext, 24
convert internal representation to or from external representation — acltotext, 24
convert a supplied name into an absolute pathname that can be used to access removable media — media_findname, 301
convert between Volume Management symbolic names, and the devices that correspond to them
— volmgt_symdev, 608
— volmgt_symname, 608
coordinate CPC library and application versions
— cpc_version, 86
copysign — return magnitude of first argument and sign of second argument, 56
cos — cosine function, 57
cosh — hyperbolic cosine function, 58
cosine function — cos, 57
cpc — hardware performance counters, 59
cpc_access — test access CPU performance counters, 62
cpc_bind_event — use CPU performance counters on lwps, 63
cpc_count_sys_events — enable and disable performance counters, 69
cpc_count_usr_events — enable and disable performance counters, 69
cpc_event — data structure to describe CPU performance counters, 71
cpc_event_accum — simple difference and accumulate operations, 73
cpc_event_diff — simple difference and accumulate operations, 73

cpc_eventtostr — translate strings to and from events, 83

cpc_getcciname — determine CPU performance counter configuration, 75

cpc_getcpuref — determine CPU performance counter configuration, 75

cpc_getnpic — determine CPU performance counter configuration, 75

cpc_getusage — determine CPU performance counter configuration, 75

cpc_pctx_bind_event — access CPU performance counters in other processes, 77

cpc_pctx_invalidate — access CPU performance counters in other processes, 77

cpc_pctx_rele — access CPU performance counters in other processes, 77

cpc_pctx_take_sample — access CPU performance counters in other processes, 77

cpc_rele — use CPU performance counters on lwps, 63

cpc_strtoevent — translate strings to and from events, 83

cpc_take_sample — use CPU performance counters on lwps, 63

cpc_version — coordinate CPC library and application versions, 86

cpc_walk_names — determine CPU performance counter configuration, 75
cplus_demangle — decode a C++ encoded symbol name, 87
create a new query —
wsreg_query_create, 627
create a new query — wsreg_query_free, 627
create audit record tokens — au_to_arg32, 35
create audit record tokens — au_to_arg64, 35
create audit record tokens — au_to_arg, 35
create audit record tokens — au_to_attr, 35
create audit record tokens — au_to_cmd, 35
create audit record tokens — au_to_data, 35
create audit record tokens — au_to_groups, 35
create audit record tokens — au_to_in_addr, 35
create audit record tokens — au_to_ipc_perm, 36
create audit record tokens — au_to_ipc, 35
create audit record tokens — au_to_iport, 35
create audit record tokens — au_to_me, 35
create audit record tokens — au_to_newgroups, 35
create audit record tokens — au_to_new_in_addr, 36
create audit record tokens — au_to_new_process, 36
create audit record tokens — au_to_new_socket, 36
create audit record tokens — au_to_new_subject, 36
create audit record tokens — au_to_opaque, 35
create audit record tokens — au_to_path, 35
create audit record tokens — au_to, 35
create audit record tokens — au_to_process, 35
create audit record tokens — au_to_process_ex, 35
create audit record tokens — au_to_return32, 35
create audit record tokens — au_to_return64, 35
create audit record tokens — au_to_return, 35
create audit record tokens — au_to_socket, 35
create audit record tokens — au_to_subject_ex, 35
create audit record tokens — au_to_subject, 35
create audit record tokens — au_to_text, 35
create or break logical connection between import and export segments —
rsms_memseg_import_connect, 501
create or break logical connection between import and export segments —
rsms_memseg_import_disconnect, 501
create or destroy barrier for imported segment —
rsms_memseg_import_destroy_barrier, 505
create or destroy barrier for imported segment —
rsms_memseg_import_init_barrier, 505
create or free local memory handle —
rsms_create_localmemory_handle, 484
create or free local memory handle —
rsms_free_localmemory_handle, 484
create or release a component —
wsreg_create_component, 624
create or release a component —
wsreg_free_component_array, 624
create or release a component —
wsreg_free_component, 624
create, destroy and manipulate exact objects —
e_alloc, 145
create, destroy and manipulate exact objects —
e_attach_to_group, 145
create, destroy and manipulate exact objects —
e_attach_to_object, 145
create, destroy and manipulate exact objects —
e_free_item, 145
create, destroy and manipulate exact objects —
e_free_object, 145
create, destroy and manipulate exact objects —
e_free, 145
create, destroy and manipulate exact objects —
e_match_object_catalog, 145
create, destroy and manipulate exact objects —
e_set_group, 145
create, destroy and manipulate exact objects —
e_set_item, 145
create, destroy and manipulate exact objects —
e_strdup, 145
create, destroy and manipulate exact objects —
e_strfree, 145
create and add node to tree and return node handle —
ptree_create_and_add_node, 455
create and add property to node and return property handle —
ptree_create_and_add_prop, 456
create DmiOctetString in dynamic memory —
newDmiOctetString, 324
create DmiString in dynamic memory —
newDmiString, 325
create handle for internal process probe control —
tnctl_internal_open, 562
create handle for kernel probe control —
tnctl_kernel_open, 564
cube root function —
cbrt, 44

data structure to describe CPU performance counters —
cpc_event, 71
decode a C++ encoded symbol name —
cplus_demangle, 87
decode a C++ encoded symbol name (continued)
— demangle, 87
demangle — decode a C++ encoded symbol name, 87
destroy a layout object —
m_destroy_layout, 300
determine CPU performance counter configuration —
cpc_getcciname, 75
determine CPU performance counter configuration —
cpc_getcpuver, 75
determine CPU performance counter configuration —
cpc_getusage, 75
determine equality of two components —
wreg_components_equal, 623
device ID interfaces for user applications —
devid_compare, 88
device ID interfaces for user applications —
devid_deviceid_to_nmlist, 88
device ID interfaces for user applications —
devid_free_nmlist, 88
device ID interfaces for user applications —
devid_free, 88
device ID interfaces for user applications —
devid_get, 88
device ID interfaces for user applications —
devid_get_minor_name, 88
device ID interfaces for user applications —
devid_getsizeof, 88
device ID interfaces for user applications —
devid_str_decode, 88
device ID interfaces for user applications —
devid_str_encode, 88
device ID interfaces for user applications —
devid_str_free, 88
device ID interfaces for user applications —
devid_valid, 88
devid_compare — device ID interfaces for user applications, 88
devid_deviceid_to_nmlist — device ID interfaces for user applications, 88
devid_free — device ID interfaces for user applications, 88

devid_free_nmlist — device ID interfaces for user applications, 88

devid_get — device ID interfaces for user applications, 88

devid_get_minor_name — device ID interfaces for user applications, 88

devid_sizeof — device ID interfaces for user applications, 88

devid_str_decode — device ID interfaces for user applications, 88

devid_str_encode — device ID interfaces for user applications, 88

devid_str_free — device ID interfaces for user applications, 88

devid_valid — device ID interfaces for user applications, 88

di_minor_devt — return libdevinfo minor node information, 100

di_minor_name — return libdevinfo minor node information, 100

di_minor_nodetype — return libdevinfo minor node information, 100

di_minor_spectype — return libdevinfo minor node information, 100

di_prom_prop_data — access PROM device information, 103

di_prom_prop_name — access PROM device information, 103

di_prom_prop_next — access PROM device information, 103

di_prop_bytes — access property values and attributes, 107

di_prop_devt — access property values and attributes, 107

di_prop_int64 — access property values and attributes, 107

di_prop_ints — access property values and attributes, 107

di_prop_lookup_bytes — search for a property, 110

di_prop_lookup_int64 — search for a property, 110

di_prop_lookup_ints — search for a property, 110

di_prop_lookup_strings — search for a property, 110

di_prop_name — access property values and attributes, 107

di_prop_strings — access property values and attributes, 107

di_prop_type — access property values and attributes, 107

di_walk_minor — traverse libdevinfo minor nodes, 114

di_walk_node — traverse libdevinfo device nodes, 116

directories
  create, remove them in a path — mkdirp, rmdirp, 308
dmi_error — print error in string form, 126
DmiAddComponent — Management Interface database administration functions, 117
DmiAddGroup — Management Interface database administration functions, 117
DmiAddLanguage — Management Interface database administration functions, 117
DmiAddRow — Management Interface operation functions, 121
DmiDeleteComponent — Management Interface database administration functions, 117
DmiDeleteGroup — Management Interface database administration functions, 117
DmiDeleteLanguage — Management Interface database administration functions, 117
DmiDeleteRow — Management Interface operation functions, 121
DmiGetAttribute — Management Interface operation functions, 121
DmiGetConfig — Management Interface initialization functions, 127
DmiGetMultiple — Management Interface operation functions, 121
DmiGetVersion — Management Interface initialization functions, 127
DmiListAttributes — Management Interface listing functions, 130
DmiListClassNames — Management Interface listing functions, 130
DmiListComponentNames — Management Interface listing functions, 130
DmiListComponentNamesByClass — Management Interface listing functions, 130
DmiListGroups — Management Interface listing functions, 130
DmiListLanguages — Management Interface listing functions, 130
DmiOriginateEvent — Service Provider functions for components, 135
DmiRegister — Management Interface initialization functions, 127
DmiRegisterCi — Service Provider functions for components, 135
DmiSetAttribute — Management Interface operation functions, 121
DmiSetConfig — Management Interface initialization functions, 127
DmiSetMultiple — Management Interface operation functions, 121
DmiUnregister — Management Interface initialization functions, 127
DmiUnRegisterCi — Service Provider functions for components, 135

ea_alloc — create, destroy and manipulate exact objects, 145
ea_attach_to_group — create, destroy and manipulate exact objects, 145
ea_attach_to_object — create, destroy and manipulate exact objects, 145
ea_close — open or close exact files, 138
ea_copy_object — construct, read, and write extended accounting records, 140
ea_copy_object_tree — construct, read, and write extended accounting records, 140
ea_error — error interface to extended accounting library, 137
ea_free — create, destroy and manipulate exact objects, 145
ea_free_item — create, destroy and manipulate exact objects, 145
ea_free_object — create, destroy and manipulate exact objects, 145
ea_get_creator — construct, read, and write extended accounting records, 140
ea_get_hostname — construct, read, and write extended accounting records, 140

elf — object file access library, 158
elf_get_entries_from_name_list — nlist, 327
elf_begin — process ELF object files, 164
elf_cntl — control an elf file descriptor, 169
elf_end — process ELF object files, 164
elf_errmsg — error handling, 171
elf_errno — error handling, 171
elf_fill — set fill byte, 172
elf_flagdata — manipulate flags, 173
elf_flagehdr — manipulate flags, 173
elf_flagelf — manipulate flags, 173
elf_flagphdr — manipulate flags, 173
elf_flagshdr — manipulate flags, 173
elf_getarhdr — retrieve archive member header, 175
elf_getarsym — retrieve archive symbol table, 177
elf_getbase — get the base offset for an object file, 178
elf_getdata — get section data, 179
elf_getident — retrieve file identification data, 184
Index 659
expm1 — computes exponential functions, 200
exponential function — exp, 199

F
fabs — absolute value function, 201
fgetprojent — project database entry
functions, 232
files
search for named file in named directories —
pathfind, 387
find node with given property and value —
pfind_node, 461
floating-point remainder value function —
mod, 203
floor — floor function, 202
floor function — floor, 202
fmod — floating-point remainder value
function, 203
free_authattr — release memory, 217
free dynamic memory allocated for input
DmiString structure
— freeDmiString, 204
free_execattr — get execution profile entry, 226
free memory for sysevent handle —
sysevent_free, 542
free_profattr — get profile description and
attributes, 230
free_proflist — get execution profile entry, 226
free_proflist — get profile description and
attributes, 230
free_userattr — get user_attr entry, 236
freeDmiString — free dynamic memory
allocated for input DmiString structure, 204
functions to manage lockfile(s) for user’s
mailbox
— maillock, 288
— mailunlock, 288
— touchlock, 288

G
gamma — log gamma function, 267
gamma_r — log gamma function, 267
get a property handle of a node —
pincl_get_first_prop, 397
get a property handle of a node —
pincl_get_next_prop, 397
get class name, subclass name, ID or buffer size
of event — sysevent_get_class_name, 544
get class name, subclass name, ID or buffer size
of event — sysevent_get_event_id, 544
get class name, subclass name, ID or buffer size
of event — sysevent_get_size, 544
get class name, subclass name, ID or buffer size
of event —
sysevent_get_subclass_name, 544
get device_deallocate entry — endddent, 222
get device_deallocate entry — getddent, 222
get device_deallocate entry — getddnam, 222
get device_deallocate entry — setddent, 222
get device_deallocate entry — setddfile, 222
get device_maps entry — enddmapent, 224
get device_maps entry — getdmapent, 224
get device_maps entry — getdmapnam, 224
get device_maps entry — getdmappdev, 224
get device_maps entry — getdmaptype, 224
get device_maps entry — getdmapent, 224
get device_maps entry — getdmapfile, 224
get execution profile entry — endexecattr, 226
get execution profile entry — free_execattr, 226
get execution profile entry — free_prolist, 226
get execution profile entry — getexecattr, 226
get execution profile entry — getexecprof, 226
get execution profile entry — getexecuser, 226
get execution profile entry — get_profiles, 226
get execution profile entry —
match_execattr, 226
get execution profile entry — setexecattr, 226
get or free interconnect topology —
rsm_free_interconnect_topology, 488
get or free interconnect topology —
rsm_get_interconnect_topology, 488
get or release a controller handle —
rsm_get_controller_attr, 486
get or release a controller handle —
rsm_get_controller, 486
get or release a poll descriptor —
rsm_memseg_get_pollfd, 500
get or release a poll descriptor —
rsm_memseg_release_pollfd, 500
get profile description and attributes —
endprofattr, 230
get profile description and attributes —
free_profattr, 230
get profile description and attributes —
free_proflist, 230
get profile description and attributes —
getprofattr, 230
get profile description and attributes —
getproflist, 230
get profile description and attributes —
getprofnam, 230
get profile description and attributes —
setprofattr, 230
get section data — elf_getdata, 179
get section data — elf_newdata, 179
get section data — elf_rawdata, 179
get the value of a property —
picl_get_propval_by_name, 403
get the value of a property —
picl_get_propval, 403
get the value of a property —
ptree_get_propval_by_name, 469
get the value of a property —
ptree_get_propval, 469
generate process audit state, 229
get audit_class database entry, 212
generate audit flags() — generate process audit state, 229
generate audit flagsbin() — convert audit flag specifications, 214
generate audit flagschar() — convert audit flag specifications, 214
generate audit_class database entry, 212
generate audit_class database entry, 212
generate audit_class database entry, 212
generate audit_class database entry, 212
get error message string — picl_strerror, 410
get handle of node specified by PICL tree path —
ptree_get_node_by_path, 464
get error message string —
picl_strerror, 410
get handle of node specified by PICL tree path —
ptree_get_node_by_path, 464
get property information —
ptree_get_propinfo, 467
get property information and handle of named property —
picl_get_propinfo_by_name, 402
get property information and handle of named property —
ptree_get_propinfo_by_name, 468
get segment ID range —
rsi_get_segmentid_range, 490
get the handle of the property by name —
picl_get_prop_by_name, 400
get the information about a property —
picl_get_propinfo, 401
get the root handle of the PICL tree —
picl_get_root, 405
get the root node handle —
pTREE_GET_ROOT, 470
get the trace attributes from a tnftcl handle —
tnftcl_trace_attrs_get, 579
getacdir — get audit control file information, 210
getacflags — get audit control file information, 210
getacinfo — get audit control file information, 210
getacflagsbin() — convert audit flag specifications, 214
getacflagschar() — convert audit flag specifications, 214
getauclassent — get audit_class database entry, 212
getauclassent_r — get audit_class database entry, 212
getauclassnam — get audit_class database entry, 212
getauclassnam_r — get audit_class database entry, 212
getauditflags() — generate process audit state, 229
getacinfo — get audit control file information, 210
getacflags — get audit control file information, 210
getacflagsbin() — convert audit flag specifications, 214
getacflagschar() — convert audit flag specifications, 214
getauclassent — get audit_class database entry, 212
getauclassent_r — get audit_class database entry, 212
getauclassnam — get audit_class database entry, 212
getauclassnam_r — get audit_class database entry, 212
getauditflags() — generate process audit state, 229
getacflagsbin() — convert audit flag specifications, 214
getacflagschar() — convert audit flag specifications, 214
getauclassent — get audit_class database entry, 212
getauclassent_r — get audit_class database entry, 212
getauevnam — get audit_event database entry, 215
getauevnam_r — get audit_event database entry, 215
getauvnonam — get audit_event database entry, 215
getauvnum — get audit_event database entry, 215
getauvnum_r — get audit_event database entry, 215
getauthattr — get authorization database entry, 217
getauthnam — get authorization database entry, 217
getauuserent — get audit_user database entry, 220
getauuserent_r — get audit_user database entry, 220
getauusernam — get audit_user database entry, 220
getauusernam_r — get audit_user database entry, 220
getddent — get device_deallocate entry, 222
getddnam — get device_deallocate entry, 222
getdefaultproj — project database entry functions, 232
getdmapent — get device_maps entry, 224
getdmapnam — get device_maps entry, 224
getdmaptdev — get device_maps entry, 224
getdmaptype — get device_maps entry, 224
getexecattr — get execution profile entry, 226
getexecprof — get execution profile entry, 226
getexecuser — get execution profile entry, 226
getprofattr — get profile description and attributes, 230
getprofalist — get profile description and attributes, 230
getprofnam — get profile description and attributes, 230
getprojbyid — project database entry functions, 232
getprojbyname — project database entry functions, 232
getprojent — project database entry functions, 232
getusercontentr — get user_attr entry, 236
getusernam — get user_attr entry, 236
getuseruid — get user_attr entry, 236

gmatch — shell global pattern matching, 238

H
hardware performance counters — cpc, 59
have Volume Management check for media — volmgmt_check, 600
hyperbolic cosine function — cosh, 58
hyperbolic sine function — sinh, 529
hyperbolic tangent function — tanh, 554
hypot — Euclidean distance function, 239

I
ilogb — returns an unbiased exponent, 240
initialize a layout object —
 m_create_layout, 296
initialize kernel statistics facility
 — kstat_close, 254
 — kstat_open, 254
initialize ptree_propinfo_t structure —
 ptree_init_propinfo, 471
initialize wsreg library — wsreg_init, 626
initiate a session with the PICL daemon —
 picl_initialize, 406
inproj — project database entry functions, 232
interfaces for direct probe and process control for another process
 — tnfctl_continue, 565
 — tnfctl_exec_open, 565
 — tnfctl_pid_open, 565
interfaces to query and to change the state of a probe
 — tnfctl_probe_connect, 573
 — tnfctl_probe_disable, 573
 — tnfctl_probe_disconnect_all, 573
 — tnfctl_probe_enable, 573
 — tnfctl_probe_state_get, 573
 — tnfctl_probe_trace, 573
 — tnfctl_probe_untrace, 573
inverse hyperbolic functions
 — acosh, 27
 — asinh, 27
 — atanh, 27
isencrypt — determine whether a buffer of characters is encrypted, 241
isnan — test for NaN, 242
iterate over probes
— tnfctl_probe_apply, 570
— tnfctl_probe_apply_ids, 570

J
j0 — Bessel functions of the first kind, 243
j1 — Bessel functions of the first kind, 243
jn — Bessel functions of the first kind, 243

K
kernel virtual memory functions
copy data from kernel image or running system — kvm_read, kvm_kread,
kvm_uread, 264
copy data to kernel image or running system — kvm_write, kvm_kwrite,
kvm_uwrite, 264
get invocation argument for process —
kvm_getcmd, 257
get u-area for process — kvm_getu, 257
get entries from kernel symbol table —
kvm_nlist, 261
kstat — kernel statistics facility, 246
kstat_chain_update — update the kstat header chain, 252
kstat_close — initialize kernel statistics facility, 254
kstat_data_lookup — find a kstat by name, 253
kstat_lookup — find a kstat by name, 253
kstat_open — initialize kernel statistics facility, 254
kstat_read — read or write kstat data, 255
kstat_write — read or write kstat data, 255
kva_match — look up a key in a key-value array, 256
kvm_close — specify kernel to examine, 262
kvm_getcmd — get invocation arguments for process, 257
kvm_getproc — read system process structures, 259
kvm_getu — get u-area for process, 257
kvm_kread — copy data from a kernel image or running system, 264
kvm_kwrite — copy data to a kernel image or running system, 264
kvm_nextproc — read system process structures, 259
kvm_nlist — get entries from kernel symbol table, 261
kvm_open — specify kernel to examine, 262
kvm_read — copy data from kernel image or running system, 264
kvm_setproc — read system process structures, 259
kvm_uread — copy data from a kernel image or running system, 264
kvm_uwrite — copy data to a kernel image or running system, 264
kvm_write — copy data to kernel image or running system, 264

L
la_activity — runtime linker auditing functions, 515
la_86_pltenter — runtime linker auditing functions, 515
la_objopen — runtime linker auditing functions, 515
la_objsearch — runtime linker auditing functions, 515
la_pltexit — runtime linker auditing functions, 515
la_pltexit64 — runtime linker auditing functions, 515
la_preinit — runtime linker auditing functions, 515
la_sparcv8_pltenter — runtime linker auditing functions, 515
la_sparcv9_pltenter — runtime linker auditing functions, 515
la_symbind32 — runtime linker auditing functions, 515
la_symbind64 — runtime linker auditing functions, 515
la_version — runtime linker auditing functions, 515
layout transformation —
   m_transform_layout, 313
layout transformation for wide character strings —
   m_wtransform_layout, 318
ld_atexit — link-editor support functions, 266
ld_atexit64 — link-editor support functions, 266
ld_file — link-editor support functions, 266
ld_file64 — link-editor support functions, 266
ld_input_done — link-editor support functions, 266
ld_input_section — link-editor support functions, 266
ld_input_section64 — link-editor support functions, 266
ld_section — link-editor support functions, 266
ld_section64 — link-editor support functions, 266
ld_start — link-editor support functions, 266
ld_start64 — link-editor support functions, 266
ld_support — link-editor support functions, 266
ld_version — link-editor support functions, 266
lgamma — log gamma function, 267
lgamma_r — log gamma function, 267
libdevinfo — library of device information functions, 269
libnvpair — library of name—value pair functions, 272
libpicl — PICL interface library, 273
libpicltree — PTree and Plug-in Registration interface library, 276
library for TNF probe control in a process or the kernel — libtnfctl, 279
library of device information functions —
   libdevinfo, 269
library of name—value pair functions —
   libnvpair, 272
libtnfctl — library for TNF probe control in a process or the kernel, 279
link-editor support functions —
   ld_atexit64, 266
link-editor support functions — ld_atexit, 266
link-editor support functions — ld_file64, 266
link-editor support functions — ld_file, 266
link-editor support functions —
   ld_input_done, 266
link-editor support functions —
   ld_input_section64, 266
link-editor support functions —
   ld_input_section, 266
link-editor support functions —
   ld_section64, 266
link-editor support functions —
   ld_section, 266
link-editor support functions —
   ld_start64, 266
link-editor support functions — ld_start, 266
link-editor support functions —
   ld_support, 266
link-editor support functions —
   ld_version, 266
load exponent of a radix-independent floating-point number — scalb, 518
load exponent of a radix-independent floating-point number — scalb, 517
log — natural logarithm function, 286
log gamma function —
   gamma, 267
   gamma_r, 267
   lgamma, 267
   lgamma_r, 267
log a message in system log — picld_log, 394
log10 — base 10 logarithm function, 284
log1p — compute natural logarithm, 285
logb — radix-independent exponent, 287

M
m_create_layout — initialize a layout object, 296
m_destroy_layout — destroy a layout object, 300
m_getvalues_layout — query layout values of a LayoutObject, 307
m_setvalues_layout — set layout values of a LayoutObject, 312
m_transform_layout — layout transformation, 313
m_wtransform_layout — layout transformation for wide character strings, 318
maillock — functions to manage lock file(s) for user’s mailbox, 288
mailunlock — functions to manage lock file(s) for user’s mailbox, 288
manage a name-value pair list —
  nvlist_alloc, 332
manage a name-value pair list —
  nvlist_dup, 332
manage a name-value pair list —
  nvlist_free, 332
manage a name-value pair list —
  nvlist_pack, 332
manage a name-value pair list —
  nvlist_size, 332
manage a name-value pair list —
  nvlist_unpack, 332
Management Interface database administration functions
  — DmiAddComponent, 117
  — DmiAddGroup, 117
  — DmiAddLanguage, 117
  — DmiDeleteComponent, 117
  — DmiDeleteGroup, 117
  — DmiDeleteLanguage, 117
Management Interface initialization functions
  — DmiGetConfig, 127
  — DmiGetVersion, 127
  — DmiRegister, 127
  — DmiSetConfig, 127
  — DmiUnregister, 127
Management Interface listing functions
  — DmiListAttributes, 130
  — DmiListClassNames, 130
  — DmiListComponents, 130
  — DmiListComponentsByClass, 130
  — DmiListGroups, 130
  — DmiListLanguages, 130
Management Interface operation functions
  — DmiAddRow, 121
  — DmiDeleteRow, 121
  — DmigetAttribute, 121
  — DmiGetMultiple, 121
  — DmiSetAttribute, 121
  — DmiSetMultiple, 121
manipulate resource pool configurations —
  pool_conf_alloc, 419
manipulate resource pool configurations —
  pool_conf_close, 419
manipulate resource pool configurations —
  pool_conf_commit, 419
manipulate resource pool configurations —
  pool_conf_export, 419
manipulate resource pool configurations —
  pool_conf_free, 419
manipulate resource pool configurations —
  pool_conf_info, 419
manipulate resource pool configurations —
  pool_conf_location, 419
manipulate resource pool configurations —
  pool_conf_open, 419
manipulate resource pool configurations —
  pool_conf_remove, 419
manipulate resource pool configurations —
  pool_conf_rollback, 419
manipulate resource pool configurations —
  pool_conf_status, 419
manipulate resource pool configurations —
  pool_conf_validate, 419
map or unmap imported segment —
  rsm_memseg_import_map, 506
map or unmap imported segment —
  rsm_memseg_import_unmap, 506
map a tnfctl error code to a string —
  tnfctl_strerror, 578
match_execattr — get execution profile entry, 226
math library exception-handling —
  matherr, 290
mathematical functions
  — gamma, 267
  — gamma_r, 267
  — lgamma, 267
  — lgamma_r, 267
matherr — math library exception-handling, 290
MD5 digest functions — md5_calc, 298  
MD5 digest functions — MD5Final, 298  
MD5 digest functions — MD5Init, 298  
MD5 digest functions — md5, 298  
MD5 digest functions — MD5Update, 298  
md5 — MD5 digest functions, 298  
MD5Final — MD5 digest functions, 298  
MD5Init — MD5 digest functions, 298  
MD5Update — MD5 digest functions, 298  
media_findname — convert a supplied name to an absolute pathname that can be used to access removable media, 301  
media_getattr — get and set media attributes, 304  
media_setattr — get and set media attributes, 304  
memory management  
copy a file into memory — copylist, 55  
mkdirp — create directories in a path, 308  
modify/delete user credentials for an authentication service — pam_setcred, 362  
mp — multiple precision integer arithmetic, 310  
mp_gcd — multiple precision integer arithmetic, 310  
mp_itom — multiple precision integer arithmetic, 310  
mp_madd — multiple precision integer arithmetic, 310  
mp_mcmp — multiple precision integer arithmetic, 310  
mp_mdiv — multiple precision integer arithmetic, 310  
mp_mfree — multiple precision integer arithmetic, 310  
mp_min — multiple precision integer arithmetic, 310  
mp_mout — multiple precision integer arithmetic, 310  
mp_msub — multiple precision integer arithmetic, 310  
mp_mtox — multiple precision integer arithmetic, 310  
mp_mult — multiple precision integer arithmetic, 310  
mp_pow — multiple precision integer arithmetic, 310  
mp_rpow — multiple precision integer arithmetic, 310  
mp_xtom — multiple precision integer arithmetic, 310  
multiple precision integer arithmetic  
— mp, 310  
— mp_gcd, 310  
— mp_itom, 310  
— mp_madd, 310  
— mp_mcmp, 310  
— mp_mdiv, 310  
— mp_mfree, 310  
— mp_min, 310  
— mp_mout, 310  
— mp_msub, 310  
— mp_mtox, 310  
— mp_mult, 310  
— mp_pow, 310  
— mp_rpow, 310  
— mp_xtom, 310  

N  
natural logarithm function — log, 286  
newDmiOctetString — create DmiOctetString in dynamic memory, 324  
newDmiString — create DmiString in dynamic memory, 325  
next representable double-precision floating-point number — nextafter, 326  
nextafter — next representable double-precision floating-point number, 326  
NOTE — annotate source code with info for tools, 328  
_NOTE — annotate source code with info for tools, 328  
NOTE — annotate source code with info for tools  
NOTE vs _NOTE, 329  
NoteInfo Argument, 329  
vlist_alloc — manage a name-value pair list, 332  
vlist_dup — manage a name-value pair list, 332  
vlist_free — manage a name-value pair list, 332
nvlist_pack — manage a name-value pair list, 332
nvlist_size — manage a name-value pair list, 332
nvlist_unpack — manage a name-value pair list, 332

O
open or close exact files — ea_close, 138
open or close exact files — ea_open, 138

P
p2close — close pipes to and from a command, 344
p2open — open pipes to and from a command, 344
PAM — PAM Service Module APIs, 368
PAM — Pluggable Authentication Module, 346
pam — Pluggable Authentication Module
  Administrative Interface, 346
  Interface Overview, 346
  Stacking Multiple Schemes, 347
  Stateful Interface, 347
pam_acct_mgmt — perform PAM account validation procedures, 349
pam_authenticate — perform authentication within the PAM framework, 350
pam_chauthtok — perform password related functions within the PAM framework, 352
pam_close_session — perform PAM session creation and termination operations, 358
pam_end — authentication transaction routines for PAM, 383
PAM error messages
  get string — pam_strerror, 386
pam_get_data — PAM routines to maintain module specific state, 364
pam_get_item — authentication information routines for PAM, 366
pam_getenv — returns the value for a PAM environment name, 354
pam_getenvlist — returns a list of all the PAM environment variables, 355
pam_open_session — perform PAM session creation and termination operations, 358
pam_putenv — change or add a value to the PAM environment, 360
PAM routines to maintain module specific state
  — pam_get_data, 364
  — pam_set_data, 364
PAM Service Module APIs
  — PAM, 368
pam_set_data — PAM routines to maintain module specific state, 364
pam_set_item — authentication information routines for PAM, 366
pam_setcred — modify/delete user credentials for an authentication service, 362
pam_sm — PAM Service Module APIs
  Interaction with the User, 369
  Interface Overview, 368
  Stateful Interface, 368
pam_sm_acct_mgmt — service provider implementation for pam_acct_mgmt, 372
pam_sm_authenticate — service provider implementation for pam_authenticate, 374
pam_sm_chauthtok — service provider implementation for pam_chauthtok, 376
pam_sm_close_session — Service provider implementation for pam_open_session and pam_close_session, 379
pam_sm_open_session — Service provider implementation for pam_open_session and pam_close_session, 379
pam_sm_setcred — service provider implementation for pam_setcred, 381
pam_start — authentication transaction routines for PAM, 383
pathfind — search for named file in named directories, 387
pctx_capture — process context library, 389
pctx_create — process context library, 389
pctx_release — process context library, 389
pctx_run — process context library, 389
pctx_set_events — associate callbacks with process events, 391
perform authentication within the PAM framework — pam_authenticate, 350
perform PAM account validation procedures — pam_acct_mgmt, 349
perform PAM session creation and termination operations
   — pam_close_session, 358
   — pam_open_session, 358
perform password related functions within the PAM framework — pam_chauthtok, 352
Perl tied hash interface to the kstat facility — Sun::Solaris::Kstat, 244
picl_get_first_prop — get a property handle of a node, 397
picl_get_next_by_col — access a table property, 399
picl_get_next_by_row — access a table property, 399
picl_get_next_prop — get a property handle of a node, 397
picl_get_prop_by_name — get the handle of the property by name, 400
picl_get_propinfo — get the information about a property, 401
picl_get_propinfo_by_name — get property information and handle of named property, 402
picl_get_propval — get the value of a property, 403
picl_get_propval_by_name — get the value of a property, 403
picl_get_root — get the root handle of the PICL tree, 405
picl_initialize — initiate a session with the PICL daemon, 406
PICL interface library — libpicl, 273
picl_set_propval — set the value of a property to the specified value, 407
picl_set_propval_by_name — set the value of a property to the specified value, 407
picl_shutdown — shutdown the session with the PICL daemon, 409
picl_strerror — get error message string, 410
picl_wait — wait for PICL tree to refresh, 411
picl_walk_tree_by_class — walk subtree by class, 412
picld_log — log a message in system log, 394
picld_plugin_register — register plug-in with the daemon, 395
pipes
   open, close to and from a command — p2open, p2close, 344
place process in new project with attendant resource controls, resource pools, and attributes — setproject, 525
Pluggable Authentication Module
   — PAM, 346
pool_associate — resource pool manipulation functions, 413
pool_component_info — resource pool component functions, 416
pool_component_to_elem — resource pool element-related functions, 418
pool_conf_alloc — manipulate resource pool configurations, 419
pool_conf_close — manipulate resource pool configurations, 419
pool_conf_commit — manipulate resource pool configurations, 419
pool_conf_export — manipulate resource pool configurations, 419
pool_conf_free — manipulate resource pool configurations, 419
pool_conf_info — manipulate resource pool configurations, 419
pool_conf_location — manipulate resource pool configurations, 419
pool_conf_open — manipulate resource pool configurations, 419
pool_conf_remove — manipulate resource pool configurations, 419
pool_conf_rollback — manipulate resource pool configurations, 419
pool_conf_status — manipulate resource pool configurations, 419
pool_conf_to_elem — resource pool element-related functions, 418
pool_conf_validate — manipulate resource pool configurations, 419
pool_create — resource pool manipulation functions, 413
pool_destroy — resource pool manipulation functions, 413
pool_dissociate — resource pool manipulation functions, 413
pool_dynamic_location — resource pool framework functions, 424
pool_error — error interface to resource pools library, 426
pool_get_binding — set and query process to resource pool bindings, 428
pool_get_owning_resource — resource pool component functions, 416
pool_get_pool — retrieve resource pool configuration elements, 431
pool_get_property — resource pool element property manipulation, 433
pool_get_resource — retrieve resource pool configuration elements, 431
pool_get_resource_binding — set and query process to resource pool bindings, 428
pool_info — resource pool manipulation functions, 413
pool_put_property — resource pool element property manipulation, 433
pool_query_components — retrieve resource pool configuration elements, 431
pool_query_pool_resources — resource pool manipulation functions, 413
pool_query_pools — retrieve resource pool configuration elements, 431
pool_query_resource_components — resource pool resource manipulation functions, 436
pool_query_resources — retrieve resource pool configuration elements, 431
pool_resource_create — resource pool resource manipulation functions, 436
pool_resource_destroy — resource pool resource manipulation functions, 436
pool_resource_info — resource pool resource manipulation functions, 436
pool_resource_to_elem — resource pool element-related functions, 418
pool_resource_transfer — resource pool resource manipulation functions, 436
pool_resource_xtransfer — resource pool resource manipulation functions, 436
pool_rm_property — resource pool element property manipulation, 433
pool_set_binding — set and query process to resource pool bindings, 428
pool_static_location — resource pool framework functions, 424
pool_strerror — error interface to resource pools library, 426
pool_to_elem — resource pool element-related functions, 418

pool_value_alloc — resource pool property value manipulation functions, 440
pool_value_free — resource pool property value manipulation functions, 440
pool_value_get_bool — resource pool property value manipulation functions, 440
pool_value_get_double — resource pool property value manipulation functions, 440
pool_value_get_int64 — resource pool property value manipulation functions, 440
pool_value_get_name — resource pool property value manipulation functions, 440
pool_value_get_string — resource pool property value manipulation functions, 440
pool_value_get_type — resource pool property value manipulation functions, 440
pool_value_get_uint64 — resource pool property value manipulation functions, 440
pool_value_set_bool — resource pool property value manipulation functions, 440
pool_value_set_double — resource pool property value manipulation functions, 440
pool_value_set_int64 — resource pool property value manipulation functions, 440
pool_value_set_name — resource pool property value manipulation functions, 440
pool_value_set_string — resource pool property value manipulation functions, 440
pool_value_set_uint64 — resource pool property value manipulation functions, 440
pool_version — resource pool framework functions, 424
pool_walk_components — walk objects within resource pool configurations, 443
pool_walk_pools — walk objects within resource pool configurations, 443
pool_walk_properties — resource pool element property manipulation, 433
pool_walk_resources — walk objects within resource pool configurations, 443
post a PICL event — ptree_post_event, 472
pow — power function, 445
power function — pow, 445
print a DmiString — printDmiString, 448
print data in DmiAttributeValues list — printDmiAttributeValues, 446
print data in input data union
   — printDmiDataUnion, 447
print error in string form
   — dmi_error, 126
printDmiAttributeValues— print data in
   DmiAttributeValues list, 446
printDmiDataUnion— print data in input data
   union, 447
printDmiString— print a DmiString, 448
probe insertion interface
   — TNF_DEBUG, 586
   — TNF_PROBE_0, 586
   — TNF_PROBE_0_DEBUG, 586
   — TNF_PROBE_1, 586
   — TNF_PROBE_1_DEBUG, 586
   — TNF_PROBE_2, 586
   — TNF_PROBE_2_DEBUG, 586
   — TNF_PROBE_3, 586
   — TNF_PROBE_3_DEBUG, 586
   — TNF_PROBE_4, 586
   — TNF_PROBE_4_DEBUG, 586
   — TNF_PROBE_5, 586
   — TNF_PROBE_5_DEBUG, 586
process context library — pctx_capture, 389
process context library — pctx_create, 389
process context library — pctx_release, 389
process context library — pctx_run, 389
project database entry functions —
   endprojent, 232
   project database entry functions —
   fgetprojent, 232
   project database entry functions —
   getdefaultproj, 232
   project database entry functions —
   getprojbyid, 232
   project database entry functions —
   getprojbyname, 232
   project database entry functions —
   getprojent, 232
   project database entry functions —
   inproj, 232
   project database entry functions —
   setprojent, 232
   project — access project files from Perl, 449
   project_walk — visit active project IDs on
   current system, 451
provide a transient program number
   — reg_ci_callback, 478
   ptree_add_node — add or delete node to or
   from tree, 453
PTree and Plug-in Registration interface library
   — libpicltree, 276
   ptree_create_and_add_node — create and add
   node to tree and return node handle, 455
   ptree_create_and_add_prop — create and add
   property to node and return property
   handle, 456
   ptree_delete_node — add or delete node to or
   from tree, 453
   ptree_find_node — find node with given
   property and value, 461
   ptree_get_node_by_path — get handle of node
   specified by PICL tree path, 464
   ptree_get_propinfo — get property
   information, 467
   ptree_get_propinfo_by_name — get property
   information and handle of named
   property, 468
   ptree_get_propval — get the value of a
   property, 469
   ptree_get_propval_by_name — get the value of
   a property, 469
   ptree_get_root — get the root node
   handle, 470
   ptree_init_propinfo — initialize
       ptree_propinfo_t structure, 471
   ptree_post_event — post a PICL event, 472
   ptree_register_handler — register a handler for
   the event, 473
   ptree_unregister_handler — unregister the
   event handler for the event, 474
   ptree_update_propval — update a property
   value, 475
   ptree_update_propval_by_name — update a
   property value, 475
   ptree_walk_tree_by_class — walk subtree by
   class, 476

Q
query layout values of a LayoutObject —
   m_getvalues_layout, 307
radix-independent exponent — logb, 287
rd_delete — runtime linker debugging functions, 516
rd_errstr — runtime linker debugging functions, 516
rd_event_addr — runtime linker debugging functions, 516
rd_event_enable — runtime linker debugging functions, 516
rd_event_getmsg — runtime linker debugging functions, 516
rd_init — runtime linker debugging functions, 516
rd_loadobj_iter — runtime linker debugging functions, 516
rd_log — runtime linker debugging functions, 516
rd_new — runtime linker debugging functions, 516
rd_objpad_enable — runtime linker debugging functions, 516
rd_plt_resolution — runtime linker debugging functions, 516
rd_reset — runtime linker debugging functions, 516
read and write a disk’s VTOC — read_vtoc, 477
read from a segment —
  rsm_memseg_import_get16, 503
read from a segment —
  rsm_memseg_import_get32, 503
read from a segment —
  rsm_memseg_import_get64, 503
read from a segment —
  rsm_memseg_import_get8, 503
read from a segment —
  rsm_memseg_import_get, 503
read system process structures —
  kvm_getproc, 259
  kvm_nextproc, 259
  kvm_setproc, 259
read and write a disk’s VTOC — read_vtoc
  write_vtoc, 477
read or write kstat data —
  kstat_read, 255
  kstat_write, 255
read_vtoc — read and write a disk’s VTOC, 477
regexp — regular expression compile and match routines, 479
register a component in the product install registry — wsreg_register, 633
register a handler for the event —
  ptree_register_handler, 473
register callbacks for probe creation and destruction —
  trncf_reg_user_funcs, 577
register plug-in with the daemon —
  picld_plugin_register, 395
regular expression compile and match routines
  — advance, 479
  — compile, 479
  — regexpr, 479
  — step, 479
release removable media device reservation —
  volmgt_release, 605
remainder — remainder function, 482
remainder function — remainder, 482
remote memory access error detection functions
  — rsm_memseg_import_close_barrier, 508
remote memory access error detection functions
  — rsm_memseg_import_open_barrier, 508
remote memory access error detection functions
  — rsm_memseg_import_order_barrier, 508
remove a component from the product install registry —
  wsreg_unregister, 647
reserve removable media device —
  volmgt_acquire, 597
resource allocation and management functions for export memory segments
  — rsm_memseg_export_create, 494
resource allocation and management functions for export memory segments
  — rsm_memseg_export_destroy, 494
resource allocation and management functions for export memory segments
  — rsm_memseg_export_rebind, 494
resource pool component functions
  — pool_component_info, 416
resource pool component functions
  — pool_get_owing_resource, 416
resource pool element property manipulation
  — pool_get_property, 433
resource pool element property manipulation
  — pool_put_property, 433
resource pool element property manipulation — pool_rm_property, 433
resource pool element property manipulation — pool_walk_properties, 433
resource pool element-related functions — pool_component_to_elem, 418
resource pool element-related functions — pool_conf_to_elem, 418
resource pool element-related functions — pool_resource_to_elem, 418
resource pool element-related functions — pool_to_elem, 418
resource pool framework functions — pool_dynamic_location, 424
resource pool framework functions — pool_static_location, 424
resource pool framework functions — pool_version, 424
resource pool manipulation functions — pool_associate, 413
resource pool manipulation functions — pool_create, 413
resource pool manipulation functions — pool_destroy, 413
resource pool manipulation functions — pool_dissociate, 413
resource pool manipulation functions — pool_info, 413
resource pool manipulation functions — pool_query_pool_resources, 413
resource pool property value manipulation functions — pool_value_alloc, 440
resource pool property value manipulation functions — pool_value_free, 440
resource pool property value manipulation functions — pool_value_get_bool, 440
resource pool property value manipulation functions — pool_value_get_double, 440
resource pool property value manipulation functions — pool_value_get_int64, 440
resource pool property value manipulation functions — pool_value_get_name, 440
resource pool property value manipulation functions — pool_value_get_string, 440
resource pool property value manipulation functions — pool_value_get_type, 440
resource pool property value manipulation functions — pool_value_get_uint64, 440
retrieve resource pool configuration elements — pool_get_pool, 431
retrieve resource pool configuration elements — pool_get_resource, 431
retrieve resource pool configuration elements — pool_query_components, 431
retrieve resource pool configuration elements — pool_query_pools, 431
retrieve resource pool configuration elements — pool_query_resources, 431
retrieve archive symbol table — elf_getarsym, 177
retrieve class-dependent object file header
  — elf32_getehdr, 150
  — elf32_newehdr, 150
  — elf64_getehdr, 150
  — elf64_newehdr, 150
retrieve class-dependent program header table
  — elf32_getphdr, 152
  — elf32_newphdr, 152
  — elf64_getphdr, 152
  — elf64_newphdr, 152
retrieve class-dependent section header
  — elf32_getshdr, 154
retrieve class-dependent section header
(continued)
— elf64_getshdr, 154
return libdevinfo minor node information —
di_minor_dev, 100
return libdevinfo minor node information —
di_minor_name, 100
return libdevinfo minor node information —
di_minor_nodetype, 100
return libdevinfo minor node information —
di_minor_spectype, 100
returns a list of all the PAM environment
variables — pam_getenclist, 355
return magnitude of first argument and sign of
second argument — copysign, 56
return the size of an object file type
— elf32_fsize, 149
— elf64_fsize, 149
returns the value for a PAM environment name
— pam_getenv, 354
return the Volume Management root directory
— volmgt_root, 606
return whether or not Volume Management is
running — volmgt_running, 607
returns an unbiased exponent — ilogb, 240
rint — round-to-nearest integral value, 483
rmdirp — remove directories in a path, 308
round-to-nearest integral value — rint, 483
rsm_create_localmemory_handle — create or
free local memory handle, 484
rsm_free_interconnect_topology — get or free
interconnect topology, 488
rsm_free_localmemory_handle — create or free
local memory handle, 484
rsm_get_controller — get or release a controller
handle, 486
rsm_get_controller_attr — get or release a
controller handle, 486
rsm_get_interconnect_topology — get or free
interconnect topology, 488
rsm_get_segmentid_range — get segment ID
range, 490
rsm_intr_signal_post — signal or wait for an
event, 492
rsm_intr_signal_wait — signal or wait for an
event, 492
rsm_memseg_export_create — resource
allocation and management functions for
export memory segments, 494
rsm_memseg_export_destroy — resource
allocation and management functions for
export memory segments, 494
rsm_memseg_export_publish — allow or
disallow a memory segment to be imported
by other nodes, 497
rsm_memseg_export_rebind — resource
allocation and management functions for
export memory segments, 494
rsm_memseg_export_republish — allow or
disallow a memory segment to be imported
by other nodes, 497
rsm_memseg_export_unpublish — allow or
disallow a memory segment to be imported
by other nodes, 497
rsm_memseg_get_pollfd — get or release a poll
descriptor, 500
rsm_memseg_import_close_barrier — remote
memory access error detection
functions, 508
rsm_memseg_import_connect — create or break
logical connection between import and
export segments, 501
rsm_memseg_import_destroy_barrier — create
or destroy barrier for imported
segment, 505
rsm_memseg_import_disconnect — create or
break logical connection between import
and export segments, 501
rsm_memseg_import_get — read from a
segment, 503
rsm_memseg_import_get_mode — set or get
mode for barrier scoping, 514
rsm_memseg_import_get16 — read from a
segment, 503
rsm_memseg_import_get32 — read from a
segment, 503
rsm_memseg_import_get64 — read from a
segment, 503
rsm_memseg_import_get8 — read from a
segment, 503
rsm_memseg_import_getv — write to a
segment using a list of I/O requests, 512
rsm_memseg_import_init_barrier — create or
destroy barrier for imported segment, 505

Index 673
rsm_memseg_import_map — map or unmap imported segment, 506
rsm_memseg_import_open_barrier — remote memory access error detection functions, 508
rsm_memseg_import_order_barrier — remote memory access error detection functions, 508
rsm_memseg_import_put — write to a segment, 510
rsm_memseg_import_put16 — write to a segment, 510
rsm_memseg_import_put32 — write to a segment, 510
rsm_memseg_import_put64 — write to a segment, 510
rsm_memseg_import_put8 — write to a segment using a list of I/O requests, 512
rsm_memseg_import_set_mode — set or get mode for barrier scoping, 514
rsm_memseg_import_unmap — map or unmap imported segment, 506
rsm_memseg_release_pollfd — get or release a poll descriptor, 500
rsm_release_controller — get or release a controller handle, 486
rtld_audit — runtime linker auditing functions, 515
rtld_db — runtime linker debugging functions, 516
runtime linker auditing functions — la_activity, 515
runtime linker auditing functions — la_i86_pltenter, 515
runtime linker auditing functions — la_sparcv8_pltenter, 515
runtime linker auditing functions — la_sparcv9_pltenter, 515
runtime linker auditing functions — la_symbind32, 515
runtime linker auditing functions — la_symbind64, 515
runtime linker auditing functions — la_version, 515
runtime linker auditing functions — rtld_audit, 515
runtime linker debugging functions — rd_delete, 516
runtime linker debugging functions — rd_errstr, 516
runtime linker debugging functions — rd_event_addr, 516
runtime linker debugging functions — rd_event_enable, 516
runtime linker debugging functions — rd_event_getmsg, 516
runtime linker debugging functions — rd_init, 516
runtime linker debugging functions — rd_loadobj_iter, 516
runtime linker debugging functions — rd_log, 516
runtime linker debugging functions — rd_new, 516
runtime linker debugging functions — rd_objpad_enable, 516
runtime linker debugging functions — rd_plt_resolution, 516
runtime linker debugging functions — rd_reset, 516
runtime linker debugging functions — rtld_db, 516

S
scalb — load exponent of a radix-independent floating-point number, 517
scalbn — load exponent of a radix-independent floating-point number, 518
search for a property — di_prop_lookup_bytes, 110
search for a property — di_prop_lookup_int64, 110
search for a property —
  
  di_prop_lookup_ints, 110

search for a property —
  
  di_prop_lookup_strings, 110

send a file — sendfile, 522

send files over sockets or copy files to files —
  sendfile, 519

sendfile — send files over sockets or copy files to files, 519

sendfile — send a file, 522

Service Provider functions for components
  — DmiOriginiateEvent, 135
  — DmiRegisterCi, 135
  — DmiUnRegisterCi, 135

service provider implementation for
  pam_acct_mgmt —
    pam_sm_acct_mgmt, 372

service provider implementation for
  pam_authenticate —
    pam_sm_authenticate, 374

service provider implementation for
  pam_chauthtok — pam_sm_chauthtok, 376

Service provider implementation for
  pam_open_session and pam_close_session
    — pam_sm_close_session, 379
    — pam_sm_open_session, 379

service provider implementation for
  pam_setcred — pam_sm_setcred, 381

set and query process to resource pool bindings —
  pool_get_binding, 428

set and query process to resource pool bindings —
  pool_get_resource_binding, 428

set and query process to resource pool bindings —
  pool_set_binding, 428

set or get mode for barrier scoping —
  rsm_memseg_import_get_mode, 514

set or get mode for barrier scoping —
  rsm_memseg_import_set_mode, 514

set or get the instance of a component —
  wsreg_get_instance, 638

set or get the instance of a component —
  wsreg_set_instance, 638

set or get the instance of a query —
  wsreg_query_get_instance, 629

set or get the instance of a query —
  wsreg_query_set_instance, 629

set or get the location of a component —
  wsreg_get_location, 640

set or get the location of a component —
  wsreg_query_get_location, 630

set or get the location of a query —
  wsreg_query_set_location, 630

set or get the parent of a component —
  wsreg_get_parent, 641

set or get the parent of a component —
  wsreg_set_parent, 641

set or get the type of a component —
  wsreg_get_type, 642

set or get the type of a component —
  wsreg_set_type, 642

set or get the uninstaller of a component —
  wsreg_get_uninstaller, 643

set or get the uninstaller of a component —
  wsreg_set_uninstaller, 643

set or get the unique name of a component —
  wsreg_get_unique_name, 644

set or get the unique name of a component —
  wsreg_set_unique_name, 644

set or get the unique name of a query —
  wsreg_query_get_unique_name, 631

set or get the unique name of a query —
  wsreg_query_set_unique_name, 631

set or get the uuid of a component —
  wsreg_get_id, 637

set or get the uuid of a component —
  wsreg_set_id, 637

set or get the uuid of a query —
  wsreg_query_get_id, 628

set or get the uuid of a query —
  wsreg_query_set_id, 628

set or get the vendor of a component —
  wsreg_get_vendor, 645

set or get the vendor of a component —
  wsreg_set_vendor, 645

set or get the version of a component —
  wsreg_get_version, 646

set or get the version of a component —
  wsreg_set_version, 646

set or get the version of a query —
  wsreg_query_get_version, 632

set or get the version of a query —
  wsreg_query_set_version, 632

set the value of a property to the specified value —
  picl_set_propval_by_name, 407
set the value of a property to the specified value  
— picl_set_propval, 407
set layout values of a LayoutObject —  
m_setvalues_layout, 312
setac — get audit control file information, 210
setauclass — rewind audit_class database file, 212
setauuser — rewind audit_event database file, 215
setauthattr — get authorization database entry, 217
setauuser — get audit_user database entry, 220
setddent — get device_deallocate entry, 222
setddfile — get device_deallocate entry, 222
setdmapent — get device_maps entry, 224
setdmapfile — get device_maps entry, 224
setexecattr — get execution profile entry, 226
setprofattr — get profile description and attributes, 230
setproject — place process in new project with  
attendant resource controls, resource pools,  
and attributes, 525
setprojent — project database entry functions, 232
setuserattr — get user_attr database entry, 236
shell global pattern matching — gmatch, 238
shutdown the session with the PICL daemon —  
picl_shutdown, 409
signal or wait for an event —  
rsm_intr_signal_post, 492
signal or wait for an event —  
rsm_intr_signal_wait, 492
significand — significand function, 527
significand function — significand, 527
simple difference and accumulate operations —  
cpc_event_accum, 73
simple difference and accumulate operations —  
cpc_event_diff, 73
sin — sine function, 528
sine function — sin, 528
sinh — hyperbolic sine function, 529
sort an ACL — acsort, 22
sqrt — square root function, 530
square root function — sqrt, 530
SSAGetTrapPort — Sun Solstice Enterprise  
Agent registration and communication  
helper functions, 531
SSAOidCmp — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidCpy — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidDup — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidFree — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidInit — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidNew — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidString — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSAOidStrToOid — Sun Solstice Enterprise  
Agent OID helper functions, 534
SSAOidZero — Sun Solstice Enterprise Agent  
OID helper functions, 534
SSARegSubagent — Sun Solstice Enterprise  
Agent registration and communication  
helper functions, 531
SSARegSubtable — Sun Solstice Enterprise  
Agent registration and communication  
helper functions, 531
SSARegSubtree — Sun Solstice Enterprise  
Agent registration and communication  
helper functions, 531
SSASendTrap — Sun Solstice Enterprise Agent  
registration and communication helper  
functions, 531
SSAStringCpy — Sun Solstice Enterprise Agent  
string helper functions, 536
SSAStringInit — Sun Solstice Enterprise Agent  
string helper functions, 536
SSAStringToChar — Sun Solstice Enterprise  
Agent string helper functions, 536
SSAStringZero — Sun Solstice Enterprise Agent  
string helper functions, 536
SSASubagentOpen — Sun Solstice Enterprise  
Agent registration and communication  
helper functions, 531
step — regular expression compile and match  
routines, 479
strcadd — copy strings, compressing or  
expanding C language escape codes, 537
strncpy — copy strings, compressing or expanding C language escape codes, 537
streadd — copy strings, compressing or expanding C language escape codes, 537
STREAMS
determine whether a buffer of characters is encrypted — isencrypt, 241
read stream up to next delimiter — bgets, 41
split buffer into fields — bufsplit, 43
sterecpy — copy strings, compressing or expanding C language escape codes, 537
string manipulations — strncmp, 539
string operations
get PAM error message string — pam_strerror, 386
strings
— strcpy, 539
— strlen, 539
— strcmp, 539
— strncmp, 539
— strcmp, 539
Sun:Solaris::Kstat — Perl tied hash interface to the kstat facility, 244
Sun Solstice Enterprise Agent OID helper functions
— SSAOidCmp, 534
— SSAOidCpy, 534
— SSAOidDup, 534
— SSAOidFree, 534
— SSAOidInit, 534
— SSAOidNew, 534
— SSAOidString, 534
— SSAOidStrToOid, 534
— SSAOidZero, 534
Sun Solstice Enterprise Agent registration and communication helper functions
— SSAgentIsAlive, 531
— SSAGetTrapPort, 531
— SSARegSubagent, 531
— SSARegSubtable, 531
— SSARegSubtree, 531
— SSASendTrap, 531
— SSASubagentOpen, 531
Sun Solstice Enterprise Agent string helper functions
— SSAStringCpy, 536
— SSAStringInit, 536
— SSAStringToChar, 536
— SSAStringZero, 536
sysevent_bind_handle — bind or unbind subscriber handle, 540
sysevent_free — free memory for sysevent handle, 542
sysevent_get_attr_list — get attribute list pointer, 543
sysevent_get_class_name — get class name, subclass name, ID or buffer size of event, 544
sysevent_get_event_id — get class name, subclass name, ID or buffer size of event, 544
sysevent_get_pid — get vendor name, publisher name or processor ID of event, 546
sysevent_get_pub_name — get vendor name, publisher name or processor ID of event, 546
sysevent_get_size — get class name, subclass name, ID or buffer size of event, 544
sysevent_get_subclass_name — get class name, subclass name, ID or buffer size of event, 544
sysevent_get_vendor_name — get vendor name, publisher name or processor ID of event, 546
sysevent_unbind_handle — bind or unbind subscriber handle, 540

T

Tan — tangent function, 553
tangent function — tan, 553
tanh — hyperbolic tangent function, 554
test access CPU performance counters — cpc_access, 62
test for NaN — isnan, 242
TNF_DEBUG — probe insertion interface, 586
TNF_PROBE — probe insertion interface
arg_name_n, 589
arg_type_n, 588
TNF_PROBE — probe insertion interface (continued)
    arg_value_n, 589
detail, 587
keys, 587
name, 587
TNF_PROBE_0 — probe insertion interface, 586
TNF_PROBE_0_DEBUG — probe insertion interface, 586
TNF_PROBE_1 — probe insertion interface, 586
TNF_PROBE_1_DEBUG — probe insertion interface, 586
TNF_PROBE_2 — probe insertion interface, 586
TNF_PROBE_2_DEBUG — probe insertion interface, 586
TNF_PROBE_3 — probe insertion interface, 586
TNF_PROBE_3_DEBUG — probe insertion interface, 586
TNF_PROBE_4 — probe insertion interface, 586
TNF_PROBE_4_DEBUG — probe insertion interface, 586
tnf_process_disable() — disables probing for the process, 591
tnf_process_enable() — enables probing for the process, 591
tnf_thread_disable() — disables probing for the calling thread, 591
tnf_thread_enable() — enables probing for the calling thread, 591
tnfctl_buffer_alloc — allocate or deallocate a buffer for trace data, 555
tnfctl_buffer_dealloc — allocate or deallocate a buffer for trace data, 555
tnfctl_checklibs — control probes of another process where caller provides /proc functionality, 559
tnfctl_close — close a tnfctl handle, 557
tnfctl_continue — interfaces for direct probe and process control for another process, 565
tnfctl_exec_open — interfaces for direct probe and process control for another process, 565
tnfctl_filter_list_add — control kernel tracing and process filtering, 581
tnfctl_filter_list_delete — control kernel tracing and process filtering, 581	nfctl_filter_list_get — control kernel tracing and process filtering, 581

678  man pages section 3: Extended Library Functions • May 2002
translate strings to and from events — cpc_eventtostr, 83
translate strings to and from events — cpc_strtoevent, 83
traverse libdevinfo device nodes — di_walk_node, 116
traverse libdevinfo minor nodes — di_walk_minor, 114

U
unregister the event handler for the event — ptree_unregister_handler, 474
update a property value —
  ptree_update_propval_by_name, 475
update a property value —
  ptree_update_propval, 475
use CPU performance counters on lwps —
  cpc_bind_event, 63
use CPU performance counters on lwps —
  cpc_rele, 63
use CPU performance counters on lwps —
  cpc_take_sample, 63

V
visit active project IDs on current system —
  project_walk, 451
volmgt_acquire — reserve removable media device, 597
volmgt_check — have Volume Management check for media, 600
volmgt_feature_enabled — check whether specific Volume Management features are enabled, 602
volmgt_inuse — check whether or not Volume Management is managing a pathname, 603
volmgt_release — release removable media device reservation, 605
volmgt_root — return the Volume Management root directory, 606
volmgt_running — return whether or not Volume Management is running, 607
volmgt_symdev — convert between Volume Management symbolic names, and the devices that correspond to them, 608
volmgt_symname — convert between Volume Management symbolic names, and the devices that correspond to them, 608
VTOC, disk’s
  read a disk’s VTOC — read_vtoc, 477
  write a disk’s VTOC — write_vtoc, 477

W
wait for PICL tree to refresh — picl_wait, 411
walk objects within resource pool configurations —
  pool_walk_components, 443
walk objects within resource pool configurations — pool_walk_pools, 443
walk objects within resource pool configurations — pool_walk_resources, 443
walk subtree by class —
  picl_walk_tree_by_class, 412
walk subtree by class —
  ptree_walk_tree_by_class, 476
write to a segment —
  rsm_memseg_import_put16, 510
write to a segment —
  rsm_memseg_import_put32, 510
write to a segment —
  rsm_memseg_import_put64, 510
write to a segment —
  rsm_memseg_import_put8, 510
write to a segment —
  rsm_memseg_import_put, 510
write to a segment using a list of I/O requests —
  rsm_memseg_import_getv, 512
write to a segment using a list of I/O requests —
  rsm_memseg_import_putv, 512
write_vtoc — read and write a disk’s VTOC, 477
wsreg_add_child_component — add or remove a child component, 610
wsreg_add_compatible_version — add or remove a backward compatible version, 612
wsreg_add_dependent_component — add or remove a dependent component, 614
wsreg_add_display_name — add, remove, or return a localized display name, 616
wsreg_add_required_component — add or remove a required component, 618
wsreg_can_access_registry — determine access to product install registry, 620
wsreg_clone_component — clone a component, 622
wsreg_components_equal — determine equality of two components, 623
wsreg_create_component — create or release a component, 624
wsreg_free_component — create or release a component, 624
wsreg_free_component_array — create or release a component, 624
wsreg_get_child_components — add or remove a child component, 610
wsreg_get_compatible_versions — add or remove a backward compatible version, 612
wsreg_get_data — add or retrieve a key-value pair, 635
wsreg_get_data_pairs — add or retrieve a key-value pair, 635
wsreg_get_dependent_components — add or remove a dependent component, 614
wsreg_get_display_languages — add, remove, or return a localized display name, 616
wsreg_get_display_name — add, remove, or return a localized display name, 616
wsreg_get_id — set or get the uuid of a component, 637
wsreg_get_instance — set or get the instance of a component, 638
wsreg_get_location — set or get the location of a component, 640
wsreg_get_parent — set or get the parent of a component, 641
wsreg_get_required_components — add or remove a required component, 618
wsreg_get_type — set or get the type of a component, 642
wsreg_get_uninstaller — set or get the uninstaller of a component, 643
wsreg_get_unique_name — set or get the unique name of a component, 644
wsreg_get_vendor — set or get the vendor of a component, 645
wsreg_get_version — set or get the version of a component, 646
wsreg_initialize — initialize wsreg library, 626
wsreg_query_create — create a new query, 627
wsreg_query_free — create a new query, 627
wsreg_query_get_id — set or get the uuid of a query, 628
wsreg_query_get_instance — set or get the instance of a query, 629
wsreg_query_get_location — set or get the location of a query, 630
wsreg_query_get_unique_name — set or get the unique name of a query, 631
wsreg_query_get_version — set or get the version of a query, 632
wsreg_query_set_id — set or get the uuid of a query, 628
wsreg_query_set_instance — set or get the instance of a query, 629
wsreg_query_set_location — set or get the location of a query, 630
wsreg_query_set_unique_name — set or get the unique name of a query, 631
wsreg_query_set_version — set or get the version of a query, 632
wsreg_remove_child_component — add or remove a child component, 610
wsreg_remove_compatible_version — add or remove a backward compatible version, 612
wsreg_remove_dependent_component — add or remove a dependent component, 614
wsreg_remove_display_name — add, remove, or return a localized display name, 616
wsreg_remove_required_component — add or remove a required component, 618
wsreg_set_data — add or retrieve a key-value pair, 635
wsreg_set_id — set or get the uuid of a component, 637
wsreg_set_instance — set or get the instance of a component, 638
wsreg_set_location — set or get the location of a component, 640
wsreg_set_parent — set or get the parent of a component, 641
wsreg_set_type — set or get the type of a component, 642
wsreg_set_uninstaller — set or get the uninstaller of a component, 643
wsreg_set_unique_name — set or get the unique name of a component, 644
wsreg_set_vendor — set or get the vendor of a component, 645
wsreg_set_version — set or get the version of a component, 646
wsreg_unregister — remove a component from the product install registry, 647

Y
y0 — Bessel functions of the second kind, 649
y1 — Bessel functions of the second kind, 649
yn — Bessel functions of the second kind, 649