man pages section 5: Standards, Environments, and Macros

Beta

Part No: 819–2252–33
February 2010

100217@23474

Contents

Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

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

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 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 9E describes the DDI (Device Driver Interface)/DKI (Driver/Kernel Interface), 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 `mtio(7I)`.
OPTIONS	This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete

	example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. 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.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

Introduction

Name Intro – introduction to miscellany

Description Among the topics presented in this section are:

Standards	The POSIX (IEEE) Standards and the X/Open Specifications are described on the `standards` page.
Environments	The user environment (`environ`), the subset of the user environment that depends on language and cultural conventions (`locale`), the large file compilation environment (`lfcompile`), and the transitional compilation environment (`lfcompile64`) are described.
Macros	The macros to format Reference Manual pages (`man` and `mansun`) as well as other text format macros (`me`, `mm`, and `ms`) are described.
Characters	Tables of character sets (`ascii`, `charmap`, `eqnchar`, and `iconv`), file format notation (`formats`), file name pattern matching (`fnmatch`), and regular expressions (`regex` and `regexp`) are presented.

Acknowledgments Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission to reproduce portions of its copyrighted documentation. Original documentation from The Open Group can be obtained online at `http://www.opengroup.org/bookstore/`.

The Institute of Electrical and Electronics Engineers and The Open Group, have given us permission to reprint portions of their documentation.

In the following statement, the phrase “this text" refers to portions of the system documentation.

Portions of this text are reprinted and reproduced in electronic form in the SunOS Reference Manual, from IEEE Std 1003.1, 2004 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright (C) 2001-2004 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between these versions and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at `http://www.opengroup.org/unix/online.html`.

This notice shall appear on any product containing this material.

REFERENCE

Standards, Environments, and Macros

Name acl – Access Control Lists

Description Access control lists (ACLs) are discretionary access control mechanisms that grant and deny access to files and directories. Two different ACL models are supported in the Solaris release:POSIX-draft ACLs and NFSv4 ACLs.

The older, POSIX-draft model is supported by the UFS file system. This model is based on a withdrawn ACL POSIX specification that was never standardized. It was subsequently withdrawn by the POSIX committee.

The other model is based on the standards of the NFSv4 working group and is an approved standard from the Internet Engineering Task Force (IETF). The ZFS file system uses the NFSv4 model, and provides richer semantics and finer grained permission capabilities than the POSIX-draft model.

POSIX-draft ACLs POSIX-draft ACLs provide an alternative security mechanism to basic UNIX file permissions in the Solaris release. Their purpose is to further restrict access to files and directories or to extend permissions to a particular user. ACLs can be used to change the permissions for the standard owner, group and other class bits of a file's mode. ACLs can give additional users and groups access to the file. A directory can also have a special kind of ACL called a *default* ACL, which defines ACL entries to be inherited by descendents of the directory. POSIX-draft ACLs have an ACL entry called *mask*. The mask defines the maximum permissions that can be granted to additional user and group entries. Whenever a file is created or its mode is changed by `chmod(1)` or `chmod(2)`, the mask is recomputed. It is recomputed to be the group permission defined in the mode passed to `chmod(2)`.

The POSIX-draft ACL model uses the standard `rwx` model of traditional UNIX permissions.

An ACL is represented as follows:

acl_entry`[,`*acl_entry*`]...`

Each *acl_entry* contains one ACL entry. An ACL entry is represented by two or three colon-separated(`:`) fields.

user:[*uid*]:*perms*	If *uid* blank, it represents the file owner.
group:[*gid*]:*perms*	If *gid* is blank, it represents the owning group.
other:*perms*	Represents the file other class.
mask:*perms*	Defines the `MAX` permission to hand out.

For example to give user `joe` read and write permissions, the ACL entry is specified as:

```
user:joe:rw-
```

NFSv4 ACLs

NFSv4 ACL model is based loosely on the Windows NT ACL model. NFSv4 ACLs provide a much richer ACL model than POSIX-draft ACLs.

The major differences between NFSv4 and POSIX-draft ACLs are as follows:

- NFSv4 ACLs provide finer grained permissions than the `rwx` model.
- NFSv4 ACLs allow for both `ALLOW` and `DENY` entries.
- NFSv4 ACLs provide a rich set of inheritance semantics. POSIX ACLs also have inheritance, but with the NFSv4 model you can control the following inheritance features:
  - Whether inheritance cascades to both files and directories or only to files or directories.
  - In the case of directories, you can indicate whether inheritance is applied to the directory itself, to just one level of subdirectories, or cascades to all subdirectories of the directory.
- NFSv4 ACLs provide a mechanism for hooking into a system's audit trail. Currently, Solaris does not support this mechanism.
- NFSv4 ACLs enable adminstrators to specify the order in which ACL entries are checked. With POSIX-draft ACLs the file system reorders ACL entries into a well defined, strict access, checking order.

POSIX-draft ACL semantics can be achieved with NFSv4 ACLs. However, only some NFSv4 ACLs can be translated to equivalent POSIX-draft ACLs.

Permissions can be specified in three different `chmod` ACL formats: verbose, compact, or positional. The verbose format uses words to indicate that the permissions are separated with a forward slash (`/`) character. Compact format uses the permission letters and positional format uses the permission letters or the hypen (`-`) to identify no permissions.

The permissions for verbose mode and their abbreviated form in parentheses for compact and positional mode are described as follows:

read_data (`r`)	Permission to read the data of the file
list_directory (`r`)	Permission to list the contents of a directory.
write_data (`w`)	Permission to modify a file's data anywhere in the file's offset range. This includes the ability to grow the file or write to any arbitrary offset.
add_file (`w`)	Permission to add a new file to a directory.
append_data (`p`)	The ability to modify the file's data, but only starting at EOF. Currently, this permission is not supported.
add_subdirectory (`p`)	Permission to create a subdirectory to a directory.

read_xattr (R)	The ability to read the extended attributes of a file or do a lookup in the extended attributes directory.
write_xattr (W)	The ability to create extended attributes or write to the extended attributes directory.
execute (x)	Permission to execute a file.
read_attributes (a)	The ability to read basic attributes (non-ACLs) of a file. Basic attributes are considered to be the stat level attributes. Allowing this access mask bit means that the entity can execute ls(1) and stat(2).
write_attributes (A)	Permission to change the times associated with a file or directory to an arbitrary value.
delete (d)	Permission to delete the file.
delete_child (D)	Permission to delete a file within a directory.
read_acl (c)	Permission to read the ACL.
write_acl (C)	Permission to write the ACL or the ability to execute chmod(1) or setfacl(1).
write_owner (o)	Permission to change the owner or the ability to execute chown(1) or chgrp(1).
synchronize (s)	Permission to access a file locally at the server with synchronous reads and writes. Currently, this permission is not supported.

The following inheritance flags are supported by NFSv4:

file_inherit (f)	Inherit to all newly created files in a directory.
dir_inherit (d)	Inherit to all newly created directories in a directory.
inherit_only (i)	Placed on a directory, but does not apply to the directory itself, only to newly created created files and directories. This flag requires file_inherit and or dir_inherit to indicate what to inherit.
no_propagate (n)	Placed on directories and indicates that ACL entries should only be inherited one level of the tree. This flag requires file_inherit and or dir_inherit to indicate what to inherit.
successful_access (S))	Indicates if an alarm or audit record should be initiated upon successful accesses. Used with audit/alarm ACE types.
failed_access (F)	Indicates if an alarm or audit record should be initiated when access fails. Used with audit/alarm ACE types.
inherited (I)	ACE was inherited.
-	No permission granted.

An NFSv4 ACL is expressed using the following syntax:

acl_entry[,*acl_entry*]...

```
owner@:<perms>[:inheritance flags]:<allow|deny>
group@:<perms>[:inheritance flags]:<allow|deny>
everyone@:<perms>[:inheritance flags]:<allow|deny>
user:<username>[:inheritance flags]:<allow|deny>
group:<groupname>[:inheritance flags]:<allow|deny>
```

owner@	File owner
group@	Group owner
user	Permissions for a specific user
group	Permissions for a specific group

Permission and inheritance flags are separated by a `/` character.

ACL specification examples:

```
user:fred:read_data/write_data/read_attributes:file_inherit:allow
owner@:read_data:allow,group@:read_data:allow,user:tom:read_data:deny
```

Using the compact ACL format, permissions are specified by using 14 unique letters to indicate permissions.

Using the positional ACL format, permissions are specified as positional arguments similar to the `ls -V` format. The hyphen (`-`), which indicates that no permission is granted at that position, can be omitted and only the required letters have to be specified.

The letters above are listed in the order they would be specified in positional notation.

With these letters you can specify permissions in the following equivalent ways.

```
user:fred:rw------R------:file_inherit:allow
```

Or you can remove the `-` and scrunch it together.

```
user:fred:rwR:file_inherit:allow
```

The inheritance flags can also be specified in a more compact manner, as follows:

```
user:fred:rwR:f:allow
user:fred:rwR:f------:allow
```

Shell-level Solaris API

The Solaris command interface supports the manipulation of ACLs. The following Solaris utilities accommodate both ACL models:

`chmod`	The `chmod` utility has been enhanced to allow for the setting and deleting of ACLs. This is achieved by extending the symbolic-mode argument to support ACL manipulation. See `chmod(1)` for details.
`compress`	When a file is compressed any ACL associated with the original file is preserved with the compressed file.
`cp`	By default, `cp` ignores ACLs, unless the `-p` option is specified. When `-p` is specified the owner and group id, permission modes, modification and access times, ACLs, and extended attributes if applicable are preserved.
`cpio`	ACLs are preserved when the `-P` option is specified.
`find`	Find locates files with ACLs when the `-acl` flag is specified.
`ls`	By default `ls` does not display ACL information. When the `-v` option is specified, a file's ACL is displayed.
`mv`	When a file is moved, all attributes are carried along with the renamed file. When a file is moved across a file system boundary, the ACLs are replicated. If the ACL information cannot be replicated, the move fails and the source file is not removed.
`pack`	When a file is packed, any ACL associated with the original file is preserved with the packed file.
`rcp`	`rcp` has been enhanced to support copying. A file's ACL is only preserved when the remote host supports ACLs.
`tar`	ACLs are preserved when the `-p` option is specified.
`unpack`	When a file with an ACL is unpacked, the unpacked file retains the ACL information.

Application-level API

The primary interfaces required to access file system ACLs at the programmatic level are the `acl_get()` and `acl_set()` functions. These functions support both POSIX draft ACLs and NFSv4 ACLs.

Retrieving a file's ACL

```
int acl_get(const char *path, int flag, acl_t **aclp);
int facl_get(int fd, int flag, acl_t **aclp);
```

The `acl_get(3SEC)` and `facl_get(3SEC)` functions retrieves an ACL on a file whose name is given by path or referenced by the open file descriptor fd. The flag argument specifies whether a trivial ACL should be retrieved. When the flag argument equals `ACL_NO_TRIVIAL` then only ACLs that are not trivial are retrieved. The ACL is returned in the `aclp` argument.

Freeing ACL structure

```
void acl_free(acl_t *aclp)s;
```

The `acl_free()` function frees up memory allocated for the argument `aclp;`.

Setting an ACL on a file

```
int acl_set(const char *path, acl_t *aclp);
int facl_set(int fd, acl_t *aclp);
```

The `acl_set(3SEC)` and `facl_get(3SEC)` functions are used for setting an ACL on a file whose name is given by path or referenced by the open file descriptor `fd`. The `aclp` argument specifies the ACL to set. The `acl_set(3SEC)` translates an POSIX-draft ACL into a NFSv4 ACL when the target file systems supports NFSv4 ACLs. No translation is performed when trying to set an NFSv4 ACL on a POSIX-draft ACL supported file system.

Determining an ACL's trivialness

```
int acl_trivial(const char *path);
```

The `acl_trivial()` function is used to determine whether a file has a trivial ACL. The trivialness of a file's ACL depends on the type of ACL it is. For POSIX-draft ACLs, it implies the ACL has greater than `MIN_ACL_ENTRIES`. For NFSv4/ZFS style ACLs, it implies that the ACL has entries other than `owner@`, `group@` and `everyone@`, inheritance flags are set, or the ACL is not ordered in a manner that meets POSIX access control requirements.

Removing all ACLs from a file

```
int acl_strip(const char *path, uid_t uid, gid_t gid, mode_t mode);
```

The `acl_strip()` function removes all ACLs from a file and replaces them with a trivial ACL based off of the passed in argument mode. After replacing the ACL the owner and group of the file are set to the values specified in the uid and gid parameters.

Converting ACLs to/from external representation

```
int acl_fromtext(const char *path, acl_t **aclp);
char *acl_totext(acl_t *aclp, int flags);
```

The `acl_text()` function converts an internal ACL representation pointed to by aclp into an external representation. See `DESCRIPTION` for details about external representation.

The `acl_fromtext()` functions converts and external representation into an internal representation. See `DESCRIPTION` for details about external representation.

Examples

The following examples demonstrate how the API can be used to perform basic operations on ACLs.

EXAMPLE 1 Retrieving and Setting an ACL

Use the following to retrieve an ACL and set it on another file:

```
error = acl_get("file", ACL_NO_TRIVIAL, &aclp);

if (error == 0 && aclp != NULL) {
error = acl_set("file2", aclp)
acl_free(aclp);
}
...
```

EXAMPLE 2 Retrieving and Setting Any ACLs

Use the following to retrieve any ACL, including trivial ACLs, and set it on another file:

```
error = acl_get("file3", 0, &aclp);
if (error == 0) {
error = acl_set("file4", aclp)
acl_free(aclp);
}
...
```

EXAMPLE 3 Determining if a File has a Trivial ACL

Use the following to determine if a file has a trivial ACL:

```
istrivial = acl_trivial("file")

if (istrivial == 0)
printf("file %s has a trivial ACL\n", file);
else
printf("file %s has a NON-trivial ACL\n", file);
...
```

EXAMPLE 4 Removing all ACLs from a File

Use the following to remove all ACLs from a file, and set a new mode, owner, and group:

```
error = acl_strip("file", 10, 100, 0644);
...
```

See Also chgrp(1), chmod(1), chown(1), cp(1), cpio(1), find(1), ls(1), mv(1), tar(1), setfacl(1), chmod(2), acl(2),stat(2),acl_get(3SEC), aclsort(3SEC), acl_fromtext(3SEC), acl_free(3SEC), acl_strip(3SEC), acl_trivial(3SEC)

Name ad – Active Directory as a naming repository

Description Solaris clients can obtain naming information from Active Directory (AD) servers.

The Solaris system must first join an AD domain and then add the `ad` keyword to the appropriate entries in the `nsswitch.conf(4)` file. The Solaris system joins the AD domain by using the `kclient(1M)` utility. The AD name service only supports the naming databases for `passwd` and `group`.

Windows users are not able to log in. The `user_attr(4)` database has no entries for Windows users, and the `passwd(1)` command does not support the synchronization of user passwords with AD.

The Solaris AD client uses auto-discovery techniques to find AD directory servers, such as domain controllers and global catalog servers. The client also uses the LDAP v3 protocol to access naming information from AD servers. The AD server schema requires no modification because the AD client works with native AD schema. The Solaris AD client uses the `idmap(1M)` service to map between Windows security identifiers (SIDs) and Solaris user identifiers (UIDs) and group identifiers (GIDs). User names and group names are taken from the `sAMAccountName` attribute of the AD user and group objects and then tagged with the domain where the objects reside. The domain name is separated from the user name or group name by the `@` character.

The client uses the SASL/GSSAPI/KRB5 security model. The `kclient` utility is used to join the client to AD. During the join operation, `kclient` configures Kerberos v5 on the client. See `kclient(1M)`.

Files

`/etc/nsswitch.conf`	Configuration file for the name-service switch.
`/etc/nsswitch.ad`	Sample configuration file for the name-service switch configured with ad, dns and files.
`/usr/lib/nss_ad.so.1`	Name service switch module for AD.

See Also `passwd(1)`, `svcs(1)`, `idmap(1M)`, `idmapd(1M)`, `kclient(1M)`, `svcadm(1M)`, `svccfg(1M)`, `svccfg(1M)`, `nsswitch.conf(4)`, `user_attr(4)`, `smf(5)`

Name ascii – map of ASCII character set

Synopsis `cat /usr/pub/ascii`

Description `/usr/pub/ascii` is a map of the ASCII character set, to be printed as needed. It contains octal and hexadecimal values for each character. While not included in that file, a chart of decimal values is also shown here.

Octal — Character

```
000 NUL  001 SOH  002 STX  003 ETX  004 EOT  005 ENQ  006 ACK  007 BEL
010 BS   011 HT   012 NL   013 VT   014 NP   015 CR   016 SO   017 SI
020 DLE  021 DC1  022 DC2  023 DC3  024 DC4  025 NAK  026 SYN  027 ETB
030 CAN  031 EM   032 SUB  033 ESC  034 FS   035 GS   036 RS   037 US
040 SP   041 !    042 "    043 #    044 $    045 %    046 &    047 ’
050 (    051 )    052 *    053 +    054 ,    055 −    056 .    057 /
060 0    061 1    062 2    063 3    064 4    065 5    066 6    067 7
070 8    071 9    072 :    073 ;    074 <    075 =    076 >    077 ?
100 @    101 A    102 B    103 C    104 D    105 E    106 F    107 G
110 H    111 I    112 J    113 K    114 L    115 M    116 N    117 O
120 P    121 Q    122 R    123 S    124 T    125 U    126 V    127 W
130 X    131 Y    132 Z    133 [    134 \    135 ]    136 ^    137 _
140 ‘    141 a    142 b    143 c    144 d    145 e    146 f    147 g
150 h    151 i    152 j    153 k    154 l    155 m    156 n    157 o
160 p    161 q    162 r    163 s    164 t    165 u    166 v    167 w
170 x    171 y    172 z    173 {    174 |    175 }    176 ~    177 DEL
```

Hexadecimal — Character

```
00 NUL   01 SOH   02 STX   03 ETX   04 EOT   05 ENQ   06 ACK  07 BEL
08 BS    09 HT    0A NL    0B VT    0C NP    0D CR    0E SO   0F SI
10 DLE   11 DC1   12 DC2   13 DC3   14 DC4   15 NAK   16 SYN  17 ETB
18 CAN   19 EM    1A SUB   1B ESC   1C FS    1D GS    1E RS   1F US
20 SP    21 !     22 "     23 #     24 $     25 %     26 &    27 ’
28 (     29 )     2A *     2B +     2C ,     2D −     2E .    2F /
30 0     31 1     32 2     33 3     34 4     35 5     36 6    37 7
38 8     39 9     3A :     3B ;     3C <     3D =     3E >    3F ?
40 @     41 A     42 B     43 C     44 D     45 E     46 F    47 G
48 H     49 I     4A J     4B K     4C L     4D M     4E N    4F O
50 P     51 Q     52 R     53 S     54 T     55 U     56 V    57 W
58 X     59 Y     5A Z     5B [     5C \     5D ]     5E ^    5F _
60 ‘     61 a     62 b     63 c     64 d     65 e     66 f    67 g
68 h     69 i     6A j     6B k     6C l     6D m     6E n    6F o
70 p     71 q     72 r     73 s     74 t     75 u     76 v    77 w
78 x     79 y     7A z     7B {     7C |     7D }     7E ~    7F DEL
```

Decimal — Character

```
  0 NUL   1 SOH    2 STX    3 ETX    4 EOT    5 ENQ    6 ACK    7 BEL
```

```
 8 BS    9 HT    10 NL    11 VT    12 NP    13 CR    14 SO    15 SI
16 DLE  17 DC1   18 DC2   19 DC3   20 DC4   21 NAK   22 SYN   23 ETB
24 CAN  25 EM    26 SUB   27 ESC   28 FS    29 GS    30 RS    31 US
32 SP   33 !     34 "     35 #     36 $     37 %     38 &     39 ’
40 (    41 )     42 *     43 +     44 ,     45 –     46 .     47 /
48 0    49 1     50 2     51 3     52 4     53 5     54 6     55 7
56 8    57 9     58 :     59 ;     60 <     61 =     62 >     63 ?
64 @    65 A     66 B     67 C     68 D     69 E     70 F     71 G
72 H    73 I     74 J     75 K     76 L     77 M     78 N     79 O
80 P    81 Q     82 R     83 S     84 T     85 U     86 V     87 W
88 X    89 Y     90 Z     91 [     92 \     93 ]     94 ^     95 _
96 ‘    97 a     98 b     99 c    100 d    101 e    102 f    103 g
104 h  105 i    106 j    107 k    108 l    109 m    110 n    111 o
112 p  113 q    114 r    115 s    116 t    117 u    118 v    119 w
120 x  121 y    122 z    123 {    124 |    125 }    126 ~    127 DEL
```

Files `/usr/pub/ascii` On-line chart of octal and hexadecimal values for the ASCII character set.

Name attributes, architecture, availability, CSI, stability, MT-Level, standard – attributes of interfaces

Description The `ATTRIBUTES` section of a manual page contains a table defining attribute types and their corresponding values. The following is an example of an attributes table. Not all attribute types are appropriate for all types of interfaces.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcsu
CSI	Enabled
Interface Stability	Committed
MT-Level	Safe
Standard	See `standards(5)`.

Architecture Architecture defines processor or specific hardware. See `-p` option of `uname(1)`. In some cases, it may indicate required adapters or peripherals.

Availability This refers to the software package which contains the command or component being described on the man page. To be able to use the command, the indicated package must have been installed. For information on how to add a package see `pkgadd(1M)`.

Code Set Independence (CSI) OS utilities and libraries free of dependencies on the properties of any code sets are said to have Code Set Independence (CSI). They have the attribute of being CSI enabled. This is in contrast to many commands and utilities, for example, that work only with Extended Unix Codesets (EUC), an encoding method that allows concurrent support for up to four code sets and is commonly used to represent Asian character sets.

For practical reasons, however, this independence is not absolute. Certain assumptions are still applied to the current CSI implementation:

- File code is a superset of ASCII.
- To support multi-byte characters and null-terminated UNIX file names, the `NULL` and `/` (slash) characters cannot be part of any multi-byte characters.
- Only “stateless” file code encodings are supported. Stateless encoding avoids shift, locking shift, designation, invocation, and so forth, although single shift is not excluded.
- Process code (`wchar_t` values) is implementation dependent and can change over time or between implementations or between locales.
- Not every object can have names composed of arbitrary characters. The names of the following objects must be composed of ASCII characters:
  - User names, group name, and passwords

- System name
- Names of printers and special devices
- Names of terminals (`/dev/tty*`)
- Process ID numbers
- Message queues, semaphores, and shared memory labels.
- The following may be composed of ISO Latin-1 or EUC characters:
  - File names
  - Directory names
  - Command names
  - Shell variables and environmental variable names
  - Mount points for file systems
  - NIS key names and domain names

- The names of NFS shared files should be composed of ASCII characters. Although files and directories may have names and contents composed of characters from non-ASCII code sets, using only the ASCII codeset allows NFS mounting across any machine, regardless of localization. For the commands and utilities that are CSI enabled, all can handle single-byte and multi-byte locales released in 2.6. For applications to get full support of internationalization services, dynamic binding has to be applied. Statically bound programs will only get support for C and POSIX locales.

Interface Stability

Sun often provides developers with early access to new technologies, which allows developers to evaluate with them as soon as possible. Unfortunately, new technologies are prone to changes and standardization often results in interface incompatibility from previous versions.

To make reasonable risk assessments, developers need to know how likely an interface is to change in future releases. To aid developers in making these assessments, interface stability information is included on some manual pages for commands, entry-points, and file formats.

The more stable interfaces can safely be used by nearly all applications, because Sun will endeavor to ensure that these continue to work in future minor releases. Applications that depend only on Committed interfaces should reliably continue to function correctly on future minor releases (but not necessarily on earlier major releases).

The less stable interfaces allow experimentation and prototyping, but should be used only with the understanding that they might change incompatibly or even be dropped or replaced with alternatives in future minor releases.

"Interfaces" that Sun does not document (for example, most kernel data structures and some symbols in system header files) may be implementation artifacts. Such internal interfaces are not only subject to incompatible change or removal, but we are unlikely to mention such a change in release notes.

Release Levels

Products are given release levels, as well as names, to aid compatibility discussions. Each release level may also include changes suitable for lower levels.

Release	Version	Significance
Major	x.0	Likely to contain major feature additions; adhere to different, possibly incompatible standard revisions; and though unlikely, could change, drop, or replace Committed interfaces. Initial product releases are usually 1.0.
Minor	x.y	Compared to an x.0 or earlier release (y!=0), it is likely to contain: feature additions, compatible changes to Committed interfaces, or likely incompatible changes to Uncommitted or Volatile interfaces.
Micro	x.y.z	Intended to be interface compatible with the previous release (z!=0), but likely to add bug fixes, performance enhancements, and support for additional hardware. Incompatible changes to Volatile interfaces are possible.

In the context of interface stability, update releases (occasionally referred to as patch releases) should be considered equivalent to Micro Releases.

Classifications

The following table summarizes how stability level classifications relate to release level. The first column lists the Stability Level. The second column lists the Release Level for Incompatible Changes, and the third column lists other comments. For a complete discussion of individual classifications, see the appropriate subsection below.

Stability	Release	Comments
Committed	Major (x.0)	Incompatibilities are exceptional.
Uncommitted	Minor (x.y)	Incompatibilities are common.
Volatile	Micro (x.y.z)	Incompatibilities are common.

The interface stability level classifications described on this manual page apply to both source and binary interfaces unless otherwise stated. All stability level classifications are public, with the exception of the `Private` classification. The precise stability level of a public interface (one that is documented in the manual pages) is unspecified unless explicitly stated. The stability level of an undocumented interface is implicitly `Private`.

The existence of documentation other than the documentation that is a component of the Solaris product should not be construed to imply any level of stability for interfaces provided by the Solaris product. The only source of stability level information is Solaris manual pages.

`Committed`
The intention of a Committed interface is to enable third parties to develop applications to these interfaces, release them, and have confidence that they will run on all releases of the product after the one in which the interface was introduced, and within the same Major release. Even at a Major release, incompatible changes are expected to be rare, and to have strong justifications.

Interfaces defined and controlled as industry standards are most often treated as Committed interfaces. In this case, the controlling body and/or public, versioned document is typically noted in a "Standard" entry in the Attributes table or elsewhere in the documentation.

Although a truly exceptional event, incompatible changes are possible in any release if the associated defect is serious enough as outlined in the Exceptions section of this document or in a Minor release by following the End of Feature process. If support of a Committed interface must be discontinued, Sun will attempt to provide notification and the stability level will be marked Obsolete.

`Uncommitted`
No commitment is made about either source or binary compatibility of these interfaces from one Minor release to the next. Even the drastic incompatible change of removal of the interface in a Minor release is possible. Uncommitted interfaces are generally not appropriate for use by release-independent products.

Incompatible changes to the interface are intended to be motivated by true improvement to the interface which may include ease of use considerations. The general expectation should be that Uncommitted interfaces are not likely to change incompatibly and if such changes occur they will be small in impact and may often have a mitigation plan.

Uncommitted interfaces generally fall into one of the following subcategorizes:

1. Interfaces that are experimental or transitional. They are typically used to give outside developers early access to new or rapidly changing technology, or to provide an interim solution to a problem where a more general solution is anticipated.
2. Interfaces whose specification is controlled by an outside body yet Sun expects to make a reasonable effort to maintain compatibility with previous releases until the next Minor release at which time Sun expects to synchronize with the external specification.
3. Interfaces whose target audience values innovation (and possibly ease of use) over stability. This attribute is often associated with administrative interfaces for higher tier components.

For Uncommitted interfaces, Sun makes no claims about either source or binary compatibility from one minor release to another. Applications developed based on these interfaces may not work in future minor releases.

`Volatile`
Volatile interfaces can change at any time and for any reason.

The Volatile interface stability level allows Sun products to quickly track a fluid, rapidly evolving specification. In many cases, this is preferred to providing additional stability to the interface, as it may better meet the expectations of the consumer.

The most common application of this taxonomy level is to interfaces that are controlled by a body other than Sun, but unlike specifications controlled by standards bodies or Free or Open Source Software (FOSS) communities which value interface compatibility, it can not be asserted that an incompatible change to the interface specification would be exceedingly rare. It may also be applied to FOSS controlled software where it is deemed more important to track the community with minimal latency than to provide stability to our customers.

It also common to apply the Volatile classification level to interfaces in the process of being defined by trusted or widely accepted organization. These are generically referred to as draft standards. An "IETF Internet draft" is a well understood example of a specification under development.

Volatile can also be applied to experimental interfaces.

No assertion is made regarding either source or binary compatibility of Volatile interfaces between any two releases, including patches. Applications containing these interfaces might fail to function properly in any future release.

`Not-an-Interface`

The situation occasionally occurs where there exists an entity that could be inferred to be an interface, but actually is not. Common examples are output from CLIs intended only for human consumption and the exact layout of a GUI.

This classification is a convenience term to be used to clarify such situations where such confusion is identified as likely. Failure to apply this term to an entity is not an indication that the entity is some form of interface. It only indicates that the potential for confusion was not identified.

`Private`

A Private interface is an interface provided by a component (or product) intended only for the use of that component. A Private interface might still be visible to or accessible by other components. Because the use of interfaces private to another component carries great stability risks, such use is explicitly not supported. Components not supplied by Sun Microsystems should not use Private interfaces.

Most Private interfaces are not documented. It is an exceptional case when a Private interface is documented. Reasons for documenting a Private interface include, but are not limited to, the intention that the interface might be reclassified to one of the public stability level classifications in the future or the fact that the interface is inordinately visible.

`Obsolete`

Obsolete is a modifier that can appear in conjunction with the above classification levels. The Obsolete modifier indicates an interface that is "deprecated" and/or no longer advised

for general use. An existing interface may be downgraded from some other status (such as Committed or Uncommitted) by the application of the Obsolete modifier to encourage customers to migrate from that interface before it may be removed (or incompatibly changed).

An Obsolete interface is supported in the current release, but is scheduled to be removed in a future (minor) release. When support of an interface is to be discontinued, Sun will attempt to provide notification before discontinuing support. Use of an Obsolete interface may produce warning messages.

Exceptions

There are rare instances when it is in the best interest of both Sun and the customer to break the interface stability commitment. The following list contains the common, known reasons for the interface provider to violate an interface stability commitment, but does not preclude others.

1. Security holes where the vulnerability is inherent in the interface.
2. Data corruption where the vulnerability is inherent in the interface.
3. Standards violations uncovered by a change in interpretation or enhancement of conformance tests.
4. An interface specification which isn't controlled by Sun has been changed incompatibly and the vast majority of interface consumers expect the newer interface.
5. Not making the incompatible change would be incomprehensible to our customers. One example of this would to have not incompatibly changed pcfs when the DOS 8.3 naming restrictions were abandoned.

Incompatible changes allowed by exception will always be delivered in the “most major” release vehicle possible. However, often the consequences of the vulnerabilities or contractual branding requirements will force delivery in a patch.

Compatibility with Earlier Interface Classification Schemes

In releases up to and including Solaris 10, a different interface classification scheme was used. The following table summarizes the mapping between the old and new classification schemes.

Old	New	Comments
Standard	Committed	An entry in the attributes table for the Standard attribute type should appear.
Stable	Committed	Name change.
Evolving	Uncommitted	Actual commitments match.
Unstable	Uncommitted	Name change.
External	Volatile	Name change with expansion of allowed usage.

Old	New	Comments
Obsolete	(Obsolete)	Was a classification, now a modifier.

The increased importance of Free or Open Source Software motivated the name change from Stable/Unstable to Committed/Uncommitted. Stable conflicted with the common use of the term in FOSS communities.

Ambiguity in the definition of Evolving was causing difficulty in interpretation. As part of the migration to the new classification scheme, many formerly Evolving interfaces were upgraded to Committed. However, upon encountering the term Evolving, Uncommitted should be inferred.

MT-Level Libraries are classified into categories that define their ability to support multiple threads. Manual pages containing functions that are of multiple or differing levels describe this in their `NOTES` or `USAGE` section.

`Safe`
: Safe is an attribute of code that can be called from a multithreaded application. The effect of calling into a Safe interface or a safe code segment is that the results are valid even when called by multiple threads. Often overlooked is the fact that the result of this Safe interface or safe code segment can have global consequences that affect all threads. For example, the action of opening or closing a file from one thread is visible by all the threads within a process. A multithreaded application has the responsibility for using these interfaces in a safe manner, which is different from whether or not the interface is Safe. For example, a multithreaded application that closes a file that is still in use by other threads within the application is not using the `close(2)` interface safely.

`Unsafe`
: An Unsafe library contains global and static data that is not protected. It is not safe to use unless the application arranges for only one thread at time to execute within the library. Unsafe libraries might contain functions that are Safe; however, most of the library's functions are unsafe to call. Some functions that are Unsafe have reentrant counterparts that are MT-Safe. Reentrant functions are designated by the `_r` suffix appended to the function name.

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

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

Async-Signal-Safe functions are also MT-Safe. Signals are disabled when locks are acquired in Async-Signal-Safe functions. These signals prevent a signal handler that might acquire the same lock from being called.

`MT-Safe with Exceptions`
See the `NOTES` or `USAGE` sections of these pages for a description of the exceptions.

`Safe with Exceptions`
See the `NOTES` or `USAGE` sections of these pages for a description of the exceptions.

`Fork-Safe`
The `fork(2)` function replicates only the calling thread in the child process. The `fork1(2)` function exists for compatibility with the past and is synonymous with `fork()`. If a thread other than the one performing the fork holds a lock when `fork()` is called, the lock will still be held in the child process but there will be no lock owner since the owning thread was not replicated. A child calling a function that attempts to acquire the lock will deadlock itself.

When `fork()` is called, a Fork-Safe library arranges to have all of its internal locks held only by the thread performing the fork. This is usually accomplished with `pthread_atfork(3C)`, which is called when the library is initialized.

The `forkall(2)` function provides the capability for the rare case when a process needs to replicate all of its threads when performing a fork. No `pthread_atfork()` actions are performed when `forkall()` is called. There are dangers associated with calling `forkall()`. If some threads in a process are performing I/O operations when another thread calls `forkall()`, they will continue performing the same I/O operations in both the parent and child processes, possibly causing data corruption. For this and other race-condition reasons, the use of `forkall()` is discouraged.

In all Solaris releases prior to Solaris 10, the behavior of `fork()` depended on whether or not the application was linked with `-lpthread` (POSIX threads, see `standards(5)`). If linked with `-lpthread`, `fork()` behaved like `fork1()`; otherwise it behaved like `forkall()`. To avoid any confusion concerning the behavior of `fork()`, applications can specifically call `fork1()` or `forkall()` as appropriate.

`Cancel-Safety`
If a multithreaded application uses `pthread_cancel(3C)` to cancel (that is, kill) a thread, it is possible that the target thread is killed while holding a resource, such as a lock or allocated memory. If the thread has not installed the appropriate cancellation cleanup handlers to release the resources appropriately (see `pthread_cancel(3C)`), the application is “cancel-unsafe”, that is, it is not safe with respect to cancellation. This unsafety could

result in deadlocks due to locks not released by a thread that gets cancelled, or resource leaks; for example, memory not being freed on thread cancellation. All applications that use `pthread_cancel(3C)` should ensure that they operate in a Cancel-Safe environment. Libraries that have cancellation points and which acquire resources such as locks or allocate memory dynamically, also contribute to the cancel-unsafety of applications that are linked with these libraries. This introduces another level of safety for libraries in a multithreaded program: Cancel-Safety. There are two sub-categories of Cancel-Safety: Deferred-Cancel-Safety, and Asynchronous-Cancel-Safety. An application is considered to be Deferred-Cancel-Safe when it is Cancel-Safe for threads whose cancellation type is `PTHREAD_CANCEL_DEFERRED`. An application is considered to be Asynchronous-Cancel-Safe when it is Cancel-Safe for threads whose cancellation type is `PTHREAD_CANCEL_ASYNCHRONOUS`. Deferred-Cancel-Safety is easier to achieve than Asynchronous-Cancel-Safety, since a thread with the deferred cancellation type can be cancelled only at well-defined cancellation points, whereas a thread with the asynchronous cancellation type can be cancelled anywhere. Since all threads are created by default to have the deferred cancellation type, it might never be necessary to worry about asynchronous cancel safety. Most applications and libraries are expected to always be Asynchronous-Cancel-Unsafe. An application which is Asynchronous-Cancel-Safe is also, by definition, Deferred-Cancel-Safe.

Standard Many interfaces are defined and controlled as industry standards. When this is the case, the controlling body and/or public, versioned document is noted in this section.

Programmers producing portable applications should rely on the interface descriptions present in the standard or specification to which the application is intended to conform, rather than the manual page descriptions of interfaces based upon a public standard. When the standard or specification allows alternative implementation choices, the manual page usually only describes the alternative implemented by Sun. The manual page also describes any compatible extensions to the base definition of Standard interfaces provided by Sun.

No endorsement of the referenced controlling body or document should be inferred by its presence as a “Standard” entry. The controlling body may be a very formal organization, as in ISO or ANSII, a less formal, but generally accepted organization such as IETF, or as informal as the sole contributor in the case of FOSS (Free or Open Source Software).

See Also `uname(1)`, `pkgadd(1M)`, `Intro(3)`, `standards(5)`

Name audit_binfile – generation of Solaris audit logs

Synopsis `/usr/lib/security/audit_binfile.so`

Description The `audit_binfile` plugin module for Solaris audit, `/usr/lib/security/audit_binfile.so`, writes binary audit data to files as configured in `audit_control(4)`; it is the default plugin for the Solaris audit daemon `auditd(1M)`. Its output is described by `audit.log(4)`.

The `audit_binfile` plugin is loaded by `auditd` if `audit_control` contains one or more lines defining audit directories by means of the `dir:` specification or if `audit_control` has a `plugin:` specification of `name=audit_binfile.so`.

Object Attributes The `p_dir` and `p_minfree` attributes are equivalent to the `dir:` and `minfree:` lines described in `audit_control`. If both the `dir:` line and the `p_dir` attribute are used, the plugin combines all directories into a single list with those specified by means of `dir:` at the front of the list. If both the `minfree` and the `p_minfree` attributes are given, the `p_minfree` value is used.

The `p_fsize` attribute defines the maximum size in bytes that an audit file can become before it is automatically closed and a new audit file opened. This is equivalent to an administrator issuing an `audit -n` command when the audit file contains the specified number of bytes. The default size is zero (0), which allows the file to grow without bound. The value specified must be within the range of [512,000, 2,147,483,647].

Examples The following directives cause `audit_binfile.so` to be loaded, specify the directories for writing audit logs, and specify the percentage of required free space per directory.

```
flags: lo,ad,-fm
naflags: lo,ad
plugin: name=audit_binfile.so;\
p_minfree=20;\
p_dir=/var/audit/jedgar/eggplant,\
/var/audit/jedgar.aux/eggplant,\
/var/audit/global/eggplant
```

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

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

See Also `auditd(1M)`, `audit_control(4)`, `syslog.conf(4)`, `attributes(5)`

System Administration Guide: Security Services

Name audit_remote – send Solaris audit logs to a remote server

Synopsis `/usr/lib/security/audit_remote.so`

Description The `audit_remote` plugin module for Solaris audit, `/usr/lib/security/audit_remote.so`, sends binary audit records (`audit.log(4)`) to audit servers specified in `audit_control(4)`.

The `audit_remote` plugin is loaded by `auditd(1M)` if `audit_control` contains a `plugin:` specification of `name=audit_remote.so`.

Object Attributes The following attributes specify the configuration of `audit_remote` plugin:

`p_hosts`

host1`[:[`*port1*`][:`*mech1*`]][,`*host2*`[:[`*port2*`][:`*mech2*`]],... \`
hostn`[:[`*portn*`][:`*mechn*`]]]`

A list of audit hosts/servers. Audit records are sent to the first available host. If a host is unreachable or a timeout occurs while sending data, the next host in the list is tried. If connection to all hosts fails, the list is tried again from the beginning.

The *host* part of a `p_hosts` entry can be in any form acceptable to `getipnodebyname(3SOCKET)`.

The *port* part of a `p_hosts` entry is the port on host that is contacted to initiate an audit server connection. If not specified, the port number is that assigned to the `solaris-audit` service. See `getservbyname(3XNET)`.

The `mech` part of a `p_host` entry is the GSS-API mechanism name (`mech(4)`). If not specified, the local host's default mechanism is used. The recommended mechanism is `kerberos_v5`.

`p_retries`
The number of retries for connecting to and sending data to a server.

The default value is `3`.

`p_timeout`
The number of seconds in which a connection/sending data timeouts.

The default value is `5` seconds.

`qsize`
The maximum number of outstanding audit records to keep.

The default is the value of the kernel queue control high water mark. See `auditconfig(1M)`.

GSS SESSION

The audit_remote plugin is a TCP client that authenticates configured audit servers using the GSS-API (libgss(3LIB)). Binary Solaris Audit records are sent with integrity and confidentiality protection as per-message tokens generated by gss_wrap(3GSS).

The plugin initiates a TCP connection to an audit server (*host:port:mech*) and establishes a GSS security context (with gss_init_sec_context(3GSS)), with appropriate security mechanism (mech(4)).

If no port is specified, the service name solaris-audit is looked up to obtain a TCP port number. If no mechanism is specified, the GSS_C_NO_OID is used as a mech_type parameter of gss_init_sec_context(3GSS), and causes the underlying GSS-API to use the local default mechanism.

gss_init_sec_context(3GSS) uses GSS_C_NO_CREDENTIAL as the initiator credential handle and a target name of the form audit@<ost_fqdn>. The server is expected to use gss_accept_sec_context(3GSS) to complete the context establishment.

Once the security context is established, the client (audit_remote plugin) calls gss_wrap(3GSS) to achieve the confidentiality of the transferred payload - the audit records. The server is expected to use gss_unwrap(3GSS) to unwrap the received data and gss_get_mic(3GSS) to obtain the MIC (Message Integrity Code) to be later sent back to the plugin as a message retrieval acknowledgment.

For example, if the kerberos_v5 mechanism is configured as GSS_API mechanism on the client and both sides agree on using this mechanism, the client side has to be eligible to non-interactively gain session keys for the audit/<host_fqdn>@<REALM> principal from the Kerberos KDC/TGS. At the same time the identity running the audit server application has to have the long term keys associated with the audit/<host_fqdn>@<REALM> principal stored in the keytab file (krb5.conf(4)) to be able to decrypt the session keys.

The audit_remote plugin initiates a connection to first server in the p_hosts list. If the connection fails or audit record sends are not responded to in p_timeout seconds, after p_retries attempts the plugin tries to connect to the next server. If the connection to the last server fails, the plugin retries to connect to the first host in the list. audit_warn(1M) is executed at every unsuccessful attempt to connect to the server or send timeout with the plugin option plugin audit_remote.so retry <count> <error>.<error> is connection <host:port> <the network error>. An EPROTO network error indicates that the client plugin did not get a successful protocol version handshake.

PROTOCOL DESCRIPTION

All protocol messages are preceded by the 4 octets of the size of the data to follow. This size is in network byte order.

The protocol begins with version negotiation followed by a GSS-API security context token exchange. On error the connection is closed (and any output token optionally sent).

The version negotiation takes place in the clear with the plugin sending an octet array of the comma (,) separated list of versions supported. The current version number is the characters

01. The receiver is expected to respond with the version that they accept (in the current case that is the characters 01). A mismatch is considered an error and the connection is closed.

The version octet array sent by the plugin and the version characters accepted by the receiver are concatenated together to make up the application data field of the channel bindings of the GSS security context establishment.

```
<plugin version characters> || <server accepted version characters>"
||" represents concatenation
```

Subsequent tokens contain a 64 bit sequence number in network byte order and a single audit record (audit.log(4)); the client uses confidentiality protection. wrap (64 bit sequence number || audit record)

The server acknowledges the receipt (and is then responsible for any data loss) with the received 64 bit sequence number and a MIC token of the unwrapped 64 bit sequence number and audit record. MIC verification on the client side acknowledges the audit record can be freed and not saved for possible retransmission.

```
64 bit sequence number || mic (64 bit sequence number || audit record)
```

Secure remote audit client/server communication flow:

```
1) Client <--> Server - TCP handshake

2) Client <--> Server - protocol version negotiation:
   a) Client  --> Server - send data size - uint32_t value (2)
   b) Client  --> Server - send clear text message of the versions
                           supported comma separated, e.g.,
                           "01,02,03" for versions 1 and 2 and 3.
                           The only version supported at present is
                           "01"
   c) Client <--  Server - send data size - uint32_t value (2)
   d) Client <--  Server - send clear text version selected
                           ("01")
   :no version match; close connection; try next host

3) Security context initiation:
   a) Client - Construct channel bindings:
         initiator address type (GSS_C_AF_NULLADDR)
         acceptor address type (GSS_C_AF_NULLADDR)
         application data value (4 octets  "0101")
   b) Client  --> Server - send token (data) size - uint32_t value
   c) Client  --> Server - GSS-API per-context token
   d) Client <--  Server - send token (data) size
   e) Client <--  Server - GSS-API per-context token
      :repeat a-e until security context is initialized; if unsuccessful,
      close connection; try next host
```

```
4) Client - transmit thread, when audit record to be sent:
   a) Client  --> Server - send data size
   b) Client  --> Server - GSS-API per-message token
                  wrap (sequence number || audit record)
      :repeat a-b while less than max (qsize) outstanding records

 5) Client - receive thread:
    a) Client <--  Server - receive data size - uint32_t value
    b) Client <--  Server - receive sequence number - uint64_t value
    c) Client <--  Server - receive MIC
    d) Client             - MIC verification - OK
    e) Client             - remove particular audit record
                            pointed by the sequence number from the
                            retransmit buffer
  :repeat a-e, on error close connection; try next host;
   retransmit unacknowledged audit records

6) Server - receive thread:
    a) Client  --> Server - receive data size
    b) Client  --> Server - GSS-API receive, uwrap, store
                   per-message token

7) Server - transmit thread:
    a) Server - MIC generation - message integrity code
                  mic (sequence number || audit record)
    b) Client <--  Server - send data size
    c) Client < -- Server - send sequence number
    d) Client <--  Server - send MIC
```

Examples

EXAMPLE 1 Loading `audit_remote.so` and Specifying the Remote Audit Servers

The following directives cause `audit_remote.so` to be loaded and specify the remote audit servers to where the audit records are sent. The `kerberos_v5` security mechanism is defined to be used when communicating with the servers.

```
plugin: name=audit_remote.so;\
p_timeout=90;p_retries=2;\
p_hosts=eggplant.eng.sun.com::kerberos_v5,\
purple.ebay.sun.com:4592:kerberos_v5
```

EXAMPLE 2 Using the Configuration of Usage Default Security Mechanism

The following example shows the configuration of usage of default security mechanism. It also shows use of default port on one of the configured servers:

```
plugin: name=audit_remote.so;\
p_timeout=10;p_retries=2;\
p_hosts=jedger.eng.sun.com,\
```

EXAMPLE 2 Using the Configuration of Usage Default Security Mechanism *(Continued)*

```
jbadams.ebay.sun.com:4592
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe
Interface Stability	See below.

The plugin configuration parameters are Committed. The client/server protocol (version "01") is Contracted Project Private. See audit.log(4) for the audit record format and content stability.

See Also

auditd(1M), auditconfig(1M), audit_warn(1M), getipnodebyname(3SOCKET), getservbyname(3XNET), gss_accept_sec_context(3GSS), gss_get_mic(3GSS), gss_init_sec_context(3GSS), gss_wrap(3GSS), gss_unwrap(3GSS), libgss(3LIB), libsocket(3LIB), audit_control(4), audit.log(4), krb5.conf(4), mech(4), attributes(5), kerberos(5), tcp(7P)

Notes

audit_remote authenticates itself to the remote audit service by way of GSS-API (libgss(3LIB)). Default gss credentials are used as provided by the gss implementation mechanism, such as Kerberos.

The solaris-audit service port assigned by IANA is 16162.

Name audit_syslog – realtime conversion of Solaris audit data to syslog messages

Synopsis `/usr/lib/security/audit_syslog.so`

Description The `audit_syslog` plugin module for Solaris audit, `/usr/lib/security/audit_syslog.so`, provides realtime conversion of Solaris audit data to syslog-formatted (text) data and sends it to a syslog daemon as configured in `syslog.conf(4)`. The plugin's path is specified in the audit configuration file, `audit_control(4)`.

Messages to `syslog` are written if selected via the `plugin` option in `audit_control`. Syslog messages are generated with the facility code of `LOG_AUDIT` (`audit` in `syslog.conf(4)`) and severity of `LOG_NOTICE`. Audit `syslog` messages contain data selected from the tokens described for the binary audit log. (See `audit.log(4)`). As with all `syslog` messages, each line in a `syslog` file consists of two parts, a `syslog` header and a message.

The syslog header contains the date and time the message was generated, the host name from which it was sent, `auditd` to indicate that it was generated by the audit daemon, an ID field used internally by `syslogd`, and `audit.notice` indicating the `syslog` facility and severity values. The `syslog` header ends with the characters `]`, that is, a closing square bracket and a space.

The message part starts with the event type from the header token. All subsequent data appears only if contained in the original audit record and there is room in the 1024-byte maximum length `syslog` line. In the following example, the backslash (`\`) indicates a continuation; actual `syslog` messages are contained on one line:

```
Oct 31 11:38:08 smothers auditd: [ID 917521 audit.notice] chdir(2) ok\
session 401 by joeuser as root:other from myultra obj /export/home
```

In the preceding example, `chdir(2)` is the event type. Following this field is additional data, described below. This data is omitted if it is not contained in the source audit record.

`ok` or `failed`	Comes from the return or exit token.
`session` <*#*>	<*#*> is the session ID from the subject token.
`by` <*name*>	<*name*> is the audit ID from the subject token.
`as` <*name*>:<*group*>	<*name*> is the effective user ID and <*group*> is the effective group ID from the subject token.
`in` <*zone name*>	The zone name. This field is generated only if the `zonename` audit policy is set.
`from` <*terminal*>	<*terminal*> is the text machine address from the subject token.
`obj` <*path*>	<*path*> is the path from the path token The path can be truncated from the left if necessary to fit it on the line. Truncation is indicated by leading ellipsis (. . .).
`proc_uid` <*owner*>	<*owner*> is the effective user ID of the process owner.

`proc_auid` <*owner*> <*owner*> is the audit ID of the process owner.

The following are example `syslog` messages:

```
Nov  4  8:27:07 smothers auditd: [ID 175219 audit.notice] \
system booted

Nov  4  9:28:17 smothers auditd: [ID 752191 audit.notice] \
login - rlogin ok session 401 by joeuser as joeuser:staff from myultra

Nov  4 10:29:27 smothers auditd: [ID 521917 audit.notice] \
access(2) ok session 255 by janeuser as janeuser:staff from  \
129.146.89.30 obj /etc/passwd
```

Object Attributes

The `p_flag` attribute, specified by means of the `plugin` directive (see `audit_control(4)`), is used to further filter audit data being sent to the `syslog` daemon beyond the classes specified through the `flags` and `naflags` lines of `audit_control` and through the user-specific lines of `audit_user(4)`. The parameter is a comma-separated list; each item represents an audit class (see `audit_class(4)`) and is specified using the same syntax used in `audit_control` for the `flags` and `naflags` lines. The default (no `p_flags` listed) is that no audit records are generated.

Examples

EXAMPLE 1 One Use of the `plugin` Line

In the specification shown below, the `plugin` line (in conjunction with `flags` and `naflags`) is used to allow class records for `lo` but allows class records for `am` for failures only. Omission of the `fm` class records results in no `fm` class records being output. The `pc` parameter has no effect because you cannot add classes to those defined by means of `flags` and `naflags` and by `audit_user(4)`. You can only remove them.

```
flags: lo,am,fm
naflags: lo
plugin: name=audit_syslog.so; p_flags=lo,-am
```

EXAMPLE 2 Use of `all`

In the specification shown below, with one exception, `all` allows all flags defined by means of `flags` and `naflags` (and `audit_user(4)`). The exception the `am` metaclass, which is equivalent to `ss,as,ua`, which is modified to output all `ua` events but only failure events for `ss` and `as`.

```
flags: lo,am
naflags: lo
plugin: name=audit_syslog.so; p_flags=all,^+ss,^+as
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe
Interface Stability	See below.

The message format and message content are Uncommitted. The configuration parameters are Committed.

See Also auditd(1M), audit_class(4), audit_control(4), syslog.conf(4), attributes(5)

System Administration Guide: Security Services

Notes Use of the `plugin` configuration line to include `audit_syslog.so` requires that `/etc/syslog.conf` is configured to store `syslog` messages of facility `audit` and severity `notice` or above in a file intended for Solaris audit records. An example of such a line in `syslog.conf` is:

```
audit.notice                /var/audit/audit.log
```

Messages from `syslog` are sent to remote `syslog` servers by means of UDP, which does not guarantee delivery or ensure the correct order of arrival of messages.

If the parameters specified for the `plugin` line result in no classes being preselected, an error is reported by means of a `syslog` alert with the `LOG_DAEMON` facility code.

The time field in the `syslog` header is generated by syslog(3C) and only approximates the time given in the binary audit log. Normally the time field shows the same whole second or at most a few seconds difference.

Name brands – alternate operating environments for non-global zones

Description The branded zone (BrandZ) framework extends the Solaris Zones infrastructure described in zones(5) to include the creation of brands, which provide non-global zones that contain non-native operating environments.

The term "brand" can refer to a wide range of operating environments. All brand management is performed as extensions to the current zones structure.

Every zone is configured with an associated brand. The brand type is used to determine which scripts are executed when a zone is installed and booted. In addition, a zone's brand is used to properly identify the correct application type at application launch time. The default brand is determined by the installed distribution in the global zone.

A branded zone will support exactly one brand of non-native binary, which means that a branded zone provides a single operating environment. Once a zone has been assigned a brand, that brand cannot be changed or removed.

BrandZ extends the zones tools in the following ways:

- A brand is an attribute of a zone, set at zone create time.
- The zonecfg tool (see zonecfg(1M)) is used to set a zone's brand type and configure the zone.
- The zoneadm tool (see zoneadm(1M)) is used to report a zone's brand type and administer the zone.

Device Support The devices supported by each zone are documented in the man pages and other documentation for that brand. The zones infrastructure detects any attempt to add an unsupported device and issues a warning to the administrator. If the administrator chooses to add an unsupported device despite that warning, that device might or might not work as expected. The configuration will be untested and unsupported.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWzoneu
Interface Stability	Committed

See Also mdb(1), zlogin(1), zonename(1), dtrace(1M), in.rlogind(1M), sshd(1M), zoneadm(1M), zonecfg(1M), kill(2), priocntl(2), getzoneid(3C), ucred_get(3C), getzoneid(3C), proc(4), attributes(5), ipkg(5), lx(5), native(5), privileges(5), solaris10(5), zones(5), lx_systrace(7D), crgetzoneid(9F)

Name cancellation – overview of concepts related to POSIX thread cancellation

Description

FUNCTION	ACTION
`pthread_cancel()`	Cancels thread execution.
`pthread_setcancelstate()`	Sets the cancellation *state* of a thread.
`pthread_setcanceltype()`	Sets the cancellation *type* of a thread.
`pthread_testcancel()`	Creates a cancellation point in the calling thread.
`pthread_cleanup_push()`	Pushes a cleanup handler routine.
`pthread_cleanup_pop()`	Pops a cleanup handler routine.

Cancellation

Thread cancellation allows a thread to terminate the execution of any application thread in the process. Cancellation is useful when further operations of one or more threads are undesirable or unnecessary.

An example of a situation that could benefit from using cancellation is an asynchronously-generated cancel condition such as a user requesting to close or exit some running operation. Another example is the completion of a task undertaken by a number of threads, such as solving a maze. While many threads search for the solution, one of the threads might solve the puzzle while the others continue to operate. Since they are serving no purpose at that point, they should all be canceled.

Planning Steps

Planning and programming for most cancellations follow this pattern:

1. Identify which threads you want to cancel, and insert `pthread_cancel(3C)` statements.
2. Identify system-defined cancellation points where a thread that might be canceled could have changed system or program state that should be restored. See the `Cancellation Points` for a list.
3. When a thread changes the system or program state just before a cancellation point, and should restore that state before the thread is canceled, place a cleanup handler before the cancellation point with `pthread_cleanup_push(3C)`. Wherever a thread restores the changed state, pop the cleanup handler from the cleanup stack with `pthread_cleanup_pop(3C)`.
4. Know whether the threads you are canceling call into cancel-unsafe libraries, and disable cancellation with `pthread_setcancelstate(3C)` before the call into the library. See `Cancellation State` and `Cancel-Safe`.
5. To cancel a thread in a procedure that contains no cancellation points, insert your own cancellation points with `pthread_testcancel(3C)`. This function creates cancellation points by testing for pending cancellations and performing those cancellations if they are found. Push and pop cleanup handlers around the cancellation point, if necessary (see Step 3, above).

Cancellation Points

The system defines certain points at which cancellation can occur (cancellation points), and you can create additional cancellation points in your application with `pthread_testcancel()`.

The following cancellation points are defined by the system (system-defined cancellation points): `creat(2)`, `aio_suspend(3C)`, `close(2)`, `creat(2)`, `getmsg(2)`, `getpmsg(2)`, `lockf(3C)`, `mq_receive(3C)`, `mq_send(3C)`, `msgrcv(2)`, `msgsnd(2)`, `msync(3C)`, `nanosleep(3C)`, `open(2)`, `pause(2)`, `poll(2)`, `pread(2)`, `pthread_cond_timedwait(3C)`, `pthread_cond_wait(3C)`, `pthread_join(3C)`, `pthread_testcancel(3C)`, `putmsg(2)`, `putpmsg(2)`, `pwrite(2)`, `read(2)`, `readv(2)`, `select(3C)`, `sem_wait(3C)`, `sigpause(3C)`, `sigwaitinfo(3C)`, `sigsuspend(2)`, `sigtimedwait(3C)`, `sigwait(2)`, `sleep(3C)`, `sync(2)`, `system(3C)`, `tcdrain(3C)`, `usleep(3C)`, `wait(3C)`, `waitid(2)`, `wait3(3C)`, `waitpid(3C)`, `write(2)`, `writev(2)`, and `fcntl(2)`, when specifying `F_SETLKW` as the command.

When cancellation is asynchronous, cancellation can occur at any time (before, during, or after the execution of the function defined as the cancellation point). When cancellation is deferred (the default case), cancellation occurs only within the scope of a function defined as a cancellation point (after the function is called and before the function returns). See `Cancellation Type` for more information about deferred and asynchronous cancellation.

Choosing where to place cancellation points and understanding how cancellation affects your program depend upon your understanding of both your application and of cancellation mechanics.

Typically, any call that might require a long wait should be a cancellation point. Operations need to check for pending cancellation requests when the operation is about to block indefinitely. This includes threads waiting in `pthread_cond_wait()` and `pthread_cond_timedwait()`, threads waiting for the termination of another thread in `pthread_join()`, and threads blocked on `sigwait()`.

A mutex is explicitly not a cancellation point and should be held for only the minimal essential time.

Most of the dangers in performing cancellations deal with properly restoring invariants and freeing shared resources. For example, a carelessly canceled thread might leave a mutex in a locked state, leading to a deadlock. Or it might leave a region of memory allocated with no way to identify it and therefore no way to free it.

Cleanup Handlers

When a thread is canceled, it should release resources and clean up the state that is shared with other threads. So, whenever a thread that might be canceled changes the state of the system or of the program, be sure to push a cleanup handler with `pthread_cleanup_push(3C)` before the cancellation point.

When a thread is canceled, all the currently-stacked cleanup handlers are executed in last-in-first-out (LIFO) order. Each handler is run in the scope in which it was pushed. When the last cleanup handler returns, the thread-specific data destructor functions are called. Thread execution terminates when the last destructor function returns.

When, in the normal course of the program, an uncanceled thread restores state that it had previously changed, be sure to pop the cleanup handler (that you had set up where the change took place) using `pthread_cleanup_pop(3C)`. That way, if the thread is canceled later, only currently-changed state will be restored by the handlers that are left in the stack.

The `pthread_cleanup_push()` and `pthread_cleanup_pop()` functions can be implemented as macros. The application must ensure that they appear as statements, and in pairs within the same lexical scope (that is, the `pthread_cleanup_push()` macro can be thought to expand to a token list whose first token is '{' with `pthread_cleanup_pop()` expanding to a token list whose last token is the corresponding '}').

The effect of the use of `return`, `break`, `continue`, and `goto` to prematurely leave a code block described by a pair of `pthread_cleanup_push()` and `pthread_cleanup_pop()` function calls is undefined.

Cancellation State

Most programmers will use only the default cancellation state of `PTHREAD_CANCEL_ENABLE`, but can choose to change the state by using `pthread_setcancelstate(3C)`, which determines whether a thread is cancelable at all. With the default *state* of `PTHREAD_CANCEL_ENABLE`, cancellation is enabled and the thread is cancelable at points determined by its cancellation *type*. See `Cancellation Type`.

If the *state* is `PTHREAD_CANCEL_DISABLE`, cancellation is disabled, the thread is not cancelable at any point, and all cancellation requests to it are held pending.

You might want to disable cancellation before a call to a cancel-unsafe library, restoring the old cancel state when the call returns from the library. See `Cancel-Safe` for explanations of cancel safety.

Cancellation Type

A thread's cancellation `type` is set with `pthread_setcanceltype(3C)`, and determines whether the thread can be canceled anywhere in its execution or only at cancellation points.

With the default *type* of `PTHREAD_CANCEL_DEFERRED`, the thread is cancelable only at cancellation points, and then only when cancellation is enabled.

If the *type* is `PTHREAD_CANCEL_ASYNCHRONOUS`, the thread is cancelable at any point in its execution (assuming, of course, that cancellation is enabled). Try to limit regions of asynchronous cancellation to sequences with no external dependencies that could result in dangling resources or unresolved state conditions. Using asynchronous cancellation is discouraged because of the danger involved in trying to guarantee correct cleanup handling at absolutely every point in the program.

Cancellation Type/State Table

Type	State	
	Enabled (Default)	Disabled

Cancellation Type/State Table		
Deferred (Default)	Cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. (Default)	All cancellation requests to the target thread are held pending.
Asynchronous	Receipt of a `pthread_cancel()` call causes immediate cancellation.	All cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executedimmediately.

Cancel-Safe

With the arrival of POSIX cancellation, the Cancel-Safe level has been added to the list of MT-Safety levels. See attributes(5). An application or library is Cancel-Safe whenever it has arranged for cleanup handlers to restore system or program state wherever cancellation can occur. The application or library is specifically Deferred-Cancel-Safe when it is Cancel-Safe for threads whose cancellation type is `PTHREAD_CANCEL_DEFERRED`. See `Cancellation State`. It is specifically Asynchronous-Cancel-Safe when it is Cancel-Safe for threads whose cancellation type is `PTHREAD_CANCEL_ASYNCHRONOUS`.

It is easier to arrange for deferred cancel safety, as this requires system and program state protection only around cancellation points. In general, expect that most applications and libraries are not Asynchronous-Cancel-Safe.

POSIX Threads Only

The cancellation functions described in this manual page are available for POSIX threads, only (the Solaris threads interfaces do not provide cancellation functions).

Examples

EXAMPLE 1 Cancellation example

The following short C++ example shows the pushing/popping of cancellation handlers, the disabling/enabling of cancellation, the use of `pthread_testcancel()`, and so on. The `free_res()` cancellation handler in this example is a dummy function that simply prints a message, but that would free resources in a real application. The function `f2()` is called from the main thread, and goes deep into its call stack by calling itself recursively.

Before `f2()` starts running, the newly created thread has probably posted a cancellation on the main thread since the main thread calls `thr_yield()` right after creating thread2. Because cancellation was initially disabled in the main thread, through a call to `pthread_setcancelstate()`, the call to `f2()` from `main()` continues and constructs X at each recursive call, even though the main thread has a pending cancellation.

When `f2()` is called for the fifty-first time (when "`i == 50`"), `f2()` enables cancellation by calling `pthread_setcancelstate()`. It then establishes a cancellation point for itself by calling `pthread_testcancel()`. (Because a cancellation is pending, a call to a cancellation point such as read(2) or write(2) would also cancel the caller here.)

EXAMPLE 1 Cancellation example *(Continued)*

After the `main()` thread is canceled at the fifty-first iteration, all the cleanup handlers that were pushed are called in sequence; this is indicated by the calls to `free_res()` and the calls to the destructor for *X*. At each level, the C++ runtime calls the destructor for *X* and then the cancellation handler, `free_res()`. The print messages from `free_res()` and *X*'s destructor show the sequence of calls.

At the end, the main thread is joined by thread2. Because the main thread was canceled, its return status from `pthread_join()` is `PTHREAD_CANCELED`. After the status is printed, thread2 returns, killing the process (since it is the last thread in the process).

```
#include <pthread.h>
#include <sched.h>
extern "C" void thr_yield(void);

extern "C" void printf(...);

struct X {
        int x;
        X(int i){x = i; printf("X(%d) constructed.\n", i);}
        ~X(){ printf("X(%d) destroyed.\n", x);}
};

void
free_res(void *i)
{
        printf("Freeing ‘%d‘\n",i);
}

char* f2(int i)
{
        try {
        X dummy(i);
        pthread_cleanup_push(free_res, (void *)i);
        if (i == 50) {
               pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
            pthread_testcancel();
        }
        f2(i+1);
        pthread_cleanup_pop(0);
        }
        catch (int) {
        printf("Error: In handler.\n");
        }
        return "f2";
}
```

EXAMPLE 1 Cancellation example *(Continued)*

```
void *
thread2(void *tid)
{
        void *sts;

        printf("I am new thread :%d\n", pthread_self());

        pthread_cancel((pthread_t)tid);

        pthread_join((pthread_t)tid, &sts);

        printf("main thread cancelled due to %d\n", sts);

        return (sts);
}

main()
{
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
        pthread_create(NULL, NULL, thread2, (void *)pthread_self());
        thr_yield();
        printf("Returned from %s\n",f2(0));
}
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also

read(2), sigwait(2), write(2), Intro(3), condition(5), pthread_cleanup_pop(3C), pthread_cleanup_push(3C), pthread_exit(3C), pthread_join(3C), pthread_setcancelstate(3C), pthread_setcanceltype(3C), pthread_testcancel(3C), setjmp(3C), attributes(5), standards(5)

Name charmap – character set description file

Description A character set description file or *charmap* defines characteristics for a coded character set. Other information about the coded character set may also be in the file. Coded character set character values are defined using symbolic character names followed by character encoding values.

The character set description file provides:

- The capability to describe character set attributes (such as collation order or character classes) independent of character set encoding, and using only the characters in the portable character set. This makes it possible to create generic `localedef(1)` source files for all codesets that share the portable character set.
- Standardized symbolic names for all characters in the portable character set, making it possible to refer to any such character regardless of encoding.

Symbolic Names Each symbolic name is included in the file and is mapped to a unique encoding value (except for those symbolic names that are shown with identical glyphs). If the control characters commonly associated with the symbolic names in the following table are supported by the implementation, the symbolic names and their corresponding encoding values are included in the file. Some of the encodings associated with the symbolic names in this table may be the same as characters in the portable character set table.

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>

Declarations The following declarations can precede the character definitions. Each must consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more blank characters, followed by the value to be assigned to the symbol.

<*code_set_name*> The name of the coded character set for which the character set description file is defined.

<*mb_cur_max*> The maximum number of bytes in a multi-byte character. This defaults to 1.

<*mb_cur_min*> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set.

<*escape_char*> The escape character used to indicate that the characters following will be interpreted in a special way, as defined later in this section. This defaults to backslash ('\'), which is the character glyph used in all the following text and examples, unless otherwise noted.

<*comment_char*> The character that when placed in column 1 of a charmap line, is used to indicate that the line is to be ignored. The default character is the number sign (#).

Format The character set mapping definitions will be all the lines immediately following an identifier line containing the string `CHARMAP` starting in column 1, and preceding a trailer line containing the string `END CHARMAP` starting in column 1. Empty lines and lines containing a <*comment_char*> in the first column will be ignored. Each non-comment line of the character set mapping definition, that is, between the `CHARMAP` and `END CHARMAP` lines of the file), must be in either of two forms:

"%s %s %s\n",<*symbolic-name*>,<*encoding*>,<*comments*>

or

"%s...%s %s %s\n",<*symbolic-name*>,<*symbolic-name*>, <*encoding*>,\
<*comments*>

In the first format, the line in the character set mapping definition defines a single symbolic name and a corresponding encoding. A character following an escape character is interpreted as itself; for example, the sequence "<\\\>>" represents the symbolic name "\>" enclosed between angle brackets.

In the second format, the line in the character set mapping definition defines a range of one or more symbolic names. In this form, the symbolic names must consist of zero or more non-numeric characters, followed by an integer formed by one or more decimal digits. The characters preceding the integer must be identical in the two symbolic names, and the integer formed by the digits in the second symbolic name must be equal to or greater than the integer formed by the digits in the first name. This is interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line must exist for all symbolic names and must define the coded character value that corresponds to the character glyph indicated in the table, or the coded character value that corresponds with the control character symbolic name. If the control characters commonly associated with the symbolic names are supported by the implementation, the symbolic name and the corresponding encoding value must be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part is expressed as one (for single-byte character values) or more concatenated decimal, octal or hexadecimal constants in the following formats:

"%cd%d",<*escape_char*>,<*decimal byte value*>

"%cx%x",<*escape_char*>,<*hexadecimal byte value*>

"%c%o",<*escape_char*>,<*octal byte value*>

Decimal Constants

Decimal constants must be represented by two or three decimal digits, preceded by the escape character and the lower-case letter d; for example, \d05, \d97, or \d143. Hexadecimal constants must be represented by two hexadecimal digits, preceded by the escape character and the lower-case letter x; for example, \x05, \x61, or \x8f. Octal constants must be represented by two or three octal digits, preceded by the escape character; for example, \05, \141, or \217. In a portable charmap file, each constant must represent an 8-bit byte. Implementations supporting other byte sizes may allow constants to represent values larger than those that can be represented in 8-bit bytes, and to allow additional digits in constants. When constants are concatenated for multi-byte character values, they must be of the same type, and interpreted in byte order from first to last with the least significant byte of the multi-byte character specified by the last constant.

Ranges of Symbolic Names

In lines defining ranges of symbolic names, the encoded value is the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range will have encoding values in increasing order. Bytes are treated as unsigned octets and carry is propagated between the bytes as necessary to represent the range. However, because this causes a null byte in the second or subsequent bytes of a character, such a declaration should not be specified. For example, the line

```
<j0101>...<j0104>     \d129\d254
```

is interpreted as:

```
<j0101>                \d129\d254
<j0102>                \d129\d255
<j0103>                \d130\d00
<j0104>                \d130\d01
```

The expanded declaration of the symbol <j0103> in the above example is an invalid specification, because it contains a null byte in the second byte of a character.

The comment is optional.

Width Specification

The following declarations can follow the character set mapping definitions (after the "END CHARMAP" statement). Each consists of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below.

WIDTH — A non-negative integer value defining the column width for the printable character in the coded character set mapping definitions. Coded

character set character values are defined using symbolic character names followed by column width values. Defining a character with more than one WIDTH produces undefined results. The END WIDTH keyword is used to terminate the WIDTH definitions. Specifying the width of a non-printable character in a WIDTH declaration produces undefined results.

WIDTH_DEFAULT — A non-negative integer value defining the default column width for any printable character not listed by one of the WIDTH keywords. If no WIDTH_DEFAULT keyword is included in the charmap, the default character width is 1.

Example:

After the "END CHARMAP" statement, a syntax for a width definition would be:

```
WIDTH
<A>             1
<B>             1
<C>...<Z>       1
...
<fool>...<foon> 2
...
END WIDTH
```

In this example, the numerical code point values represented by the symbols <A> and <B> are assigned a width of 1. The code point values < C> to <Z> inclusive, that is, <C>, <D>, <E>, and so on, are also assigned a width of 1. Using <A>. . .<Z> would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword WIDTH_DEFAULT could have been added as appropriate.

See Also locale(1), localedef(1), nl_langinfo(3C), extensions(5), locale(5)

Name condition – concepts related to condition variables

Description Occasionally, a thread running within a mutex needs to wait for an event, in which case it blocks or sleeps. When a thread is waiting for another thread to communicate its disposition, it uses a condition variable in conjunction with a mutex. Although a mutex is exclusive and the code it protects is sharable (at certain moments), condition variables enable the synchronization of differing events that share a mutex, but not necessarily data. Several condition variables may be used by threads to signal each other when a task is complete, which then allows the next waiting thread to take ownership of the mutex.

A condition variable enables threads to atomically block and test the condition under the protection of a mutual exclusion lock (mutex) until the condition is satisfied. If the condition is false, a thread blocks on a condition variable and atomically releases the mutex that is waiting for the condition to change. If another thread changes the condition, it may wake up waiting threads by signaling the associated condition variable. The waiting threads, upon awakening, reacquire the mutex and re-evaluate the condition.

Initialize Condition variables and mutexes should be global. Condition variables that are allocated in writable memory can synchronize threads among processes if they are shared by the cooperating processes (see `mmap(2)`) and are initialized for this purpose.

The scope of a condition variable is either intra-process or inter-process. This is dependent upon whether the argument is passed implicitly or explicitly to the initialization of that condition variable. A condition variable does not need to be explicitly initialized. A condition variable is initialized with all zeros, by default, and its scope is set to within the calling process. For inter-process synchronization, a condition variable must be initialized once, and only once, before use.

A condition variable must not be simultaneously initialized by multiple threads or re-initialized while in use by other threads.

Condition variables attributes may be set to the default or customized at initialization. POSIX threads even allow the default values to be customized. Establishing these attributes varies depending upon whether POSIX or Solaris threads are used. Similar to the distinctions between POSIX and Solaris thread creation, POSIX condition variables implement the default, intra-process, unless an attribute object is modified for inter-process prior to the initialization of the condition variable. Solaris condition variables also implement as the default, intra-process; however, they set this attribute according to the argument, *type*, passed to their initialization function.

Condition Wait The condition wait interface allows a thread to wait for a condition and atomically release the associated mutex that it needs to hold to check the condition. The thread waits for another thread to make the condition true and that thread's resulting call to signal and wakeup the waiting thread.

Condition Signaling

A condition signal allows a thread to unblock the next thread waiting on the condition variable, whereas, a condition broadcast allows a thread to unblock all threads waiting on the condition variable.

Destroy

The condition destroy functions destroy any state, but not the space, associated with the condition variable.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also

`fork(2)`, `mmap(2)`, `setitimer(2)`, `shmop(2)`, `cond_broadcast(3C)`, `cond_destroy(3C)`, `cond_init(3C)`, `cond_signal(3C)`, `cond_timedwait(3C)`, `cond_wait(3C)`, `pthread_cond_broadcast(3C)`, `pthread_cond_destroy(3C)`, `pthread_cond_init(3C)`, `pthread_cond_signal(3C)`, `pthread_cond_timedwait(3C)`, `pthread_cond_wait(3C)`, `pthread_condattr_init(3C)`, `signal(3C)`, `attributes(5)`, `mutex(5)`, `standards(5)`

Notes

If more than one thread is blocked on a condition variable, the order in which threads are unblocked is determined by the scheduling policy.

`USYNC_THREAD` does not support multiple mapplings to the same logical synch object. If you need to `mmap()` a synch object to different locations within the same address space, then the synch object should be initialized as a shared object `USYNC_PROCESS` for Solaris, and `PTHREAD_PROCESS_PRIVATE` for POSIX.

Name crypt_bsdbf – password hashing module using Blowfish cryptographic algorithm

Synopsis `/usr/lib/security/$ISA/crypt_bsdbf.so`

Description The `crypt_bsdbf` module is a one-way password hashing module for use with `crypt(3C)` that uses the Blowfish cryptographic algorithm. The algorithm identifier for `crypt.conf(4)` and `policy.conf(4)` is `2a`.

The maximum password length for `crypt_bsdbf` is 72 characters.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`

Name crypt_bsdmd5 – password hashing module using MD5 message hash algorithm

Synopsis

```
/usr/lib/security/$ISA/crypt_bsdmd5.so
```

Description The `crypt_bsdmd5` module is a one-way password hashing module for use with `crypt(3C)` that uses the MD5 message hash algorithm. The algorithm identifier for `crypt.conf(4)` and `policy.conf(4)` is `1`. The output is compatible with `md5crypt` on BSD and Linux systems.

The maximum password length for `crypt_bsdmd5` is 255 characters.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`

Name crypt_sha256 – password hashing module using SHA–256 message hash algorithm

Synopsis `/usr/lib/security/$ISA/crypt_sha256.so`

Description The `crypt_sha256` module is a one-way password hashing module for use with `crypt(3C)` that uses the SHA–256 message hash algorithm. The algorithm identifier for `crypt.conf(4)` and `policy.conf(4)` is `5`.

This module is designed to make it difficult to crack passwords that use brute force attacks based on high speed SHA–256 implementations that use code inlining, unrolled loops, and table lookup.

The maximum password length for `crypt_sha256` is 255 characters.

The following options can be passed to the module by means of `crypt.conf(4)`:

`rounds=`<*positive_number*>
Specifies the number of rounds of SHA-256 to use in generation of the salt; the default number of rounds is 5000. Negative values have no effect and are ignored. The minimum number of rounds cannot be below 1000.

The number of additional rounds is stored in the salt string returned by `crypt_gensalt(3C)`. For example:

```
$5,rounds=6000$nlxmTTpz$
```

When `crypt_gensalt(3C)` is being used to generate a new salt, if the number of additional rounds configured in `crypt.conf(4)` is greater than that in the old salt, the value from `crypt.conf(4)` is used instead. This allows for migration to stronger (but more time-consuming) salts on password change.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`

Name crypt_sha512 – password hashing module using SHA–512 message hash algorithm

Synopsis

```
/usr/lib/security/$ISA/crypt_sha512.so
```

Description The `crypt_sha512` module is a one-way password hashing module for use with `crypt(3C)` that uses the SHA–512 message hash algorithm. The algorithm identifier for `crypt.conf(4)` and `policy.conf(4)` is 6.

This module is designed to make it difficult to crack passwords that use brute force attacks based on high speed SHA–512 implementations that use code inlining, unrolled loops, and table lookup.

The maximum password length for `crypt_sha512` is 255 characters.

The following options can be passed to the module by means of `crypt.conf(4)`:

`rounds=`<*positive_number*>
: Specifies the number of rounds of SHA-512 to use in generation of the salt; the default number of rounds is 5000. Negative values have no effect and are ignored. The minimum number of rounds cannot be below 1000.

  The number of additional rounds is stored in the salt string returned by `crypt_gensalt(3C)`. For example:

  ```
  $6,rounds=6000$nlxmTTpz$
  ```

  When `crypt_gensalt(3C)` is being used to generate a new salt, if the number of additional rounds configured in `crypt.conf(4)` is greater than that in the old salt, the value from `crypt.conf(4)` is used instead. This allows for migration to stronger (but more time-consuming) salts on password change.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`

Name crypt_sunmd5 – password hashing module using MD5 message hash algorithm

Synopsis `/usr/lib/security/$ISA/crypt_sunmd5.so`

Description The `crypt_sunmd5` module is a one-way password hashing module for use with `crypt(3C)` that uses the MD5 message hash algorithm. The algorithm identifier for `crypt.conf(4)` and `policy.conf(4)` is `md5`.

This module is designed to make it difficult to crack passwords that use brute force attacks based on high speed MD5 implementations that use code inlining, unrolled loops, and table lookup.

The maximum password length for `crypt_sunmd5` is 255 characters.

The following options can be passed to the module by means of `crypt.conf(4)`:

`rounds=`<*positive_number*>
: Specifies the number of additional rounds of MD5 to use in generation of the salt; the default number of rounds is 4096. Negative values have no effect and are ignored, that is, the number of rounds cannot be lowered below 4096.

  The number of additional rounds is stored in the salt string returned by `crypt_gensalt(3C)`. For example:

  ```
  $md5,rounds=1000$nlxmTTpz$
  ```

  When `crypt_gensalt(3C)` is being used to generate a new salt, if the number of additional rounds configured in `crypt.conf(4)` is greater than that in the old salt, the value from `crypt.conf(4)` is used instead. This allows for migration to stronger (but more time-consuming) salts on password change.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`

Name crypt_unix – traditional UNIX crypt algorithm

Description The `crypt_unix` algorithm is the traditional UNIX crypt algorithm. It is not considered sufficiently secure for current systems and is provided for backwards compatibility. The `crypt_sunmd5(5)`, `crypt_bsdmd5(5)`, or `crypt_bsdbf(5)` algorithm should be used instead.

The algorithm identifier for `policy.conf(4)` is `__unix__`. There is no entry in `crypt.conf(4)` for this algorithm.

The `crypt_unix` algorithm is internal to `libc` and provides the string encoding function used by `crypt(3C)` when the first character of the salt is not a "$".

This algorithm is based on a one-way encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search. Only the first eight characters of the key passed to `crypt()` are used with this algorithm; the rest are silently ignored. The salt is a two-character string chosen from the set [a-zA-Z0-9./]. This string is used to perturb the hashing algorithm in one of 4096 different ways.

The maximum password length for `crypt_unix` is 8 characters.

Usage The return value of the `crypt_unix` algorithm might not be portable among standard-conforming systems. See `standards(5)`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

See Also `passwd(1)`, `crypt(3C)`, `crypt_genhash_impl(3C)`, `crypt_gensalt(3C)`, `crypt_gensalt_impl(3C)`, `getpassphrase(3C)`, `crypt.conf(4)`, `passwd(4)`, `policy.conf(4)`, `attributes(5)`, `crypt_bsdbf(5)`, `crypt_bsdmd5(5)`, `crypt_sunmd5(5)`, `standards(5)`

Name device_clean – device clean programs

Description Each allocatable device has a device clean program associated with it. Device clean programs are invoked by `deallocate(1)` to clean device states, registers, and any residual information in the device before the device is allocated to a user. Such cleaning is required by the object reuse policy.

Use `list_devices(1)` to obtain the names and types of allocatable devices as well as the cleaning program and the authorizations that are associated with each device.

On a system configured with Trusted Extensions, device clean programs are also invoked by `allocate(1)`, in which case the program can optionally mount appropriate media for the caller.

The following device clean programs reside in `/etc/security/lib`.

`audio_clean`	audio devices
`fd_clean`	floppy devices
`st_clean`	tape devices
`sr_clean`	CD-ROM devices

On a system configured with Trusted Extensions, the following additional cleaning programs and wrappers are available.

`disk_clean`	floppy, CD-ROM, and other removable media devices. This program mounts the device during the execution of `allocate`, if required.
`audio_clean_wrapper`	wrapper to make audio_clean work with CDE
`wdwwrapper`	wrapper to make other cleaning programs work with CDE
`wdwmsg`	CDE dialog boxes for cleaning programs

Administrators can create device clean programs for their sites. These programs must adhere to the syntax described below.

`/etc/security/lib/`*device-clean-program* `[–i | –f | –s | –I] \`
`–m` *mode* `–u` *user-name* `–z` *zone-name* `–p` *zone-path* *device-name*

where:

device-name	The name of the device that is to be cleaned. Use `list_devices` to obtain the list of allocatable devices.
`-i`	Invoke boot-time initialization.
`-f`	Force cleanup by the administrator.
`-s`	Invoke standard cleanup by the user.

`-I` Same as `-i`, with no error or warning.

The following options are supported only when the system is configured with Trusted Extensions.

`-m` *mode* Specify the mode in which the clean program is invoked. Valid values are allo- cate and deallocate. The default mode is allocate.

`-u` *user-name* Specify the name of user who executes the device clean program. The default user is the caller.

`-z` *zone-name* Specify the name of the zone in which the device is to be allocated or deallocated. The default zone is the global zone.

`-p` *zone-path* Establish the root path of the zone that is specified by *zone-name*. Default is "/".

Exit Status The following exit values are returned:

`0`
Successful completion.

`1`
An error. Caller can place device in error state.

`2`
A system error. Caller can place device in error state.

On a system configured with Trusted Extensions, the following additional exit values are returned:

`3`
Mounting of device failed. Caller shall not place device in error state.

`4`
Mounting of device succeeded.

Files `/etc/security/lib/*` device clean programs

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	See below.

The Invocation is Uncommitted. The Output is Not-an-interface.

See Also `allocate`(1), `deallocate`(1), `list_devices`(1), `attributes`(5)

System Administration Guide: Security Services

Name dhcp – Dynamic Host Configuration Protocol

Description Dynamic Host Configuration Protocol (DHCP) enables host systems in a TCP/IP network to be configured automatically for the network as they boot. DHCP uses a client/server mechanism: servers store configuration information for clients, and provide that information upon a client's request. The information can include the client's IP address and information about network services available to the client.

This manual page provides a brief summary of the Solaris DHCP implementation.

Solaris DHCP Client

The Solaris DHCP client is implemented as background daemon, `dhcpagent(1M)`.

For IPv4, this daemon is started automatically during bootup if there exists at least one `dhcp.`*interface* file in `/etc`. Only interfaces with a corresponding `/etc/dhcp.`*interface* file are automatically configured during boot.

For IPv6, this daemon is started automatically when commanded by `in.ndpd` (based on IPv6 Routing Advertisement messages). No `/etc/dhcp.`*interface* file is necessary, but such a file can be used to specify an interface as "primary," provided that IPv4 DHCP is also in use.

Network parameters needed for system configuration during bootup are extracted from the information received by the daemon through the use of the `dhcpinfo(1)` command. The daemon's default behavior can be altered by changing the tunables in the `/etc/default/dhcpagent` file. The daemon is controlled by the `ifconfig(1M)` utility. Check the status of the daemon using the `netstat(1M)` and `ifconfig(1M)` commands.

Solaris DHCP Server

The Solaris DHCP server is implemented as a background daemon, `in.dhcpd(1M)`. This daemon can deliver network configuration information to either `BOOTP` or DHCP clients. The Solaris DHCP service can be managed using the `dhcpmgr(1M)` GUI or the command line utilities `dhcpconfig(1M)`, `dhtadm(1M)`, and `pntadm(1M)`.

DHCP Configuration Tables

The Solaris DHCP server stores client configuration information in the following two types of tables:

`dhcptab` tables — Contain macros and options (also known as symbols), used to construct a package of configuration information to send to each DHCP client. There exists only one `dhcptab` for the DHCP service. The `dhcptab(4)` can be viewed and modified using the `dhtadm(1M)` command or `dhcpmgr(1M)` graphical utility. See `dhcptab(4)` for more information about the syntax of `dhcptab` records. See `dhcp_inittab(4)` for more information about the DHCP options and symbols.

DHCP network tables — DHCP network tables, which contain mappings of client IDs to IP addresses and parameters associated with those addresses. Network tables are named with the IP address of the network, and can be created, viewed, and modified using the `pntadm` command or

`dhcpmgr` graphical utility. See `dhcp_network(4)` for more information about network tables.

See Also `dhcpinfo(1)`, `dhcpagent(1M)`, `dhcpconfig(1M)`, `dhcpmgr(1M)`, `dhtadm(1M)`, `ifconfig(1M)`, `in.dhcpd(1M)`, `in.ndpd(1M)`, `netstat(1M)`, `pntadm(1M)`, `syslog(3C)`, `dhcp_network(4)`, `dhcptab(4)`, `dhcpsvc.conf(4)`, `dhcp_inittab(4)`, `ndpd.conf(4)`, `dhcp_modules(5)`

Alexander, S., and R. Droms. *RFC 2132, DHCP Options and BOOTP Vendor Extensions*. Silicon Graphics, Inc. Bucknell University. March 1997.

Droms, R. *RFC 1534, Interoperation Between DHCP and BOOTP*. Bucknell University. October 1993.

Droms, R. *RFC 2131, Dynamic Host Configuration Protocol*. Bucknell University. March 1997.

Wimer, W. *RFC 1542, Clarifications and Extensions for the Bootstrap Protocol*. Carnegie Mellon University. October 1993.

Lemon, T. and B. Sommerfeld. *RFC 4361, Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)*. Nominum and Sun Microsystems. February 2006.

Droms, R. *RFC 3315, Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. Cisco Systems. July 2003.

Name dhcp_modules – data storage modules for the DHCP service

Description This man page describes the characteristics of data storage modules (public modules) for use by the Solaris Dynamic Host Configuration Protocol (DHCP) service.

Public modules are the part of the DHCP service architecture that encapsulate the details of storing DHCP service data in a data storage service. Examples of data storage services are Oracle and `ufs` file systems.

Public modules are dynamic objects which can be shipped separately from the Solaris DHCP service. Once installed, a public module is visible to the DHCP service, and can be selected for use by the service through the DHCP service management interfaces (`dhcpmgr(1M)`, `dhcpconfig(1M)`, `dhtadm(1M)`, and `pntadm(1M)`).

Public modules may be provided by Sun Microsystems, Inc or by third parties.

The Solaris DHCP service management architecture provides a mechanism for plugging in public module-specific administration functionality into the `dhcpmgr`(1M) and `dhcpconfig`(1M) utilities. This functionality is in the form of a Java Bean, which is provided by the public module vendor. This Java Bean collects public module-specific configuration from the user (you) and provides it to the Solaris DHCP service.

The Solaris DHCP service bundles three modules with the service, which are described below. There are three `dhcpsvc.conf`(4) DHCP service configuration parameters pertaining to public modules: `RESOURCE`, `PATH`, and `RESOURCE_CONFIG`. See `dhcpsvc.conf(4)` for more information about these parameters.

SUNWfiles This module stores its data in ASCII files. Although the format is ASCII, hand-editing is discouraged. It is useful for DHCP service environments that support several hundred to a couple thousand of clients and lease times are a few hours or more.

This module's data may be shared between DHCP servers through the use of NFS.

SUNWbinfiles This module stores its data in binary files. It is useful for DHCP service environments with many networks and many thousands of clients. This module provides an order of magnitude increase in performance and capacity over SUNWfiles.

This module's data cannot be shared between DHCP servers.

See Also `crontab(1)`, `dhcpconfig(1M)`, `dhcpmgr(1M)`, `dhtadm(1M)`, `pntadm(1M)`, `dhcpsvc.conf(4)`, `dhcp(5)`

Name environ – user environment

Description When a process begins execution, one of the exec family of functions makes available an array of strings called the environment; see exec(2). By convention, these strings have the form *variable=value*, for example, PATH=/sbin:/usr/sbin. These environmental variables provide a way to make information about a program's environment available to programs.

A name may be placed in the environment by the export command and *name=value* arguments in sh(1), or by one of the exec functions. It is unwise to conflict with certain shell variables such as MAIL, PS1, PS2, and IFS that are frequently exported by .profile files; see profile(4).

The following environmental variables can be used by applications and are expected to be set in the target run-time environment.

HOME
: The name of the user's login directory, set by login(1) from the password file; see passwd(4).

LANG
: The string used to specify internationalization information that allows users to work with different national conventions. The setlocale(3C) function checks the LANG environment variable when it is called with "" as the locale argument. LANG is used as the default locale if the corresponding environment variable for a particular category is unset or null. If, however, LC_ALL is set to a valid, non-empty value, its contents are used to override both the LANG and the other LC_* variables. For example, when invoked as setlocale(LC_CTYPE, ""), setlocale() will query the LC_CTYPE environment variable first to see if it is set and non-null. If LC_CTYPE is not set or null, then setlocale() will check the LANG environment variable to see if it is set and non-null. If both LANG and LC_CTYPE are unset or NULL, the default "C" locale will be used to set the LC_CTYPE category.

  Most commands will invoke setlocale(LC_ALL, "") prior to any other processing. This allows the command to be used with different national conventions by setting the appropriate environment variables.

  The following environment variables correspond to each category of setlocale(3C):

  LC_ALL
  : If set to a valid, non-empty string value, override the values of LANG and all the other LC_*variables.

  LC_COLLATE
  : This category specifies the character collation sequence being used. The information corresponding to this category is stored in a database created by the localedef(1) command. This environment variable affects strcoll(3C) and strxfrm(3C).

LC_CTYPE
: This category specifies character classification, character conversion, and widths of multibyte characters. When LC_CTYPE is set to a valid value, the calling utility can display and handle text and file names containing valid characters for that locale; Extended Unix Code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide; and EUC characters of 1, 2, or 3 column widths. The default "C" locale corresponds to the 7-bit ASCII character set; only characters from ISO 8859-1 are valid. The information corresponding to this category is stored in a database created by the localedef() command. This environment variable is used by ctype(3C), mblen(3C), and many commands, such as cat(1), ed(1), ls(1), and vi(1).

LC_MESSAGES
: This category specifies the language of the message database being used. For example, an application may have one message database with French messages, and another database with German messages. Message databases are created by the mkmsgs(1) command. This environment variable is used by exstr(1), gettxt(1), srchtxt(1), gettxt(3C), and gettext(3C).

LC_MONETARY
: This category specifies the monetary symbols and delimiters used for a particular locale. The information corresponding to this category is stored in a database created by the localedef(1) command. This environment variable is used by localeconv(3C).

LC_NUMERIC
: This category specifies the decimal and thousands delimiters. The information corresponding to this category is stored in a database created by the localedef() command. The default C locale corresponds to "." as the decimal delimiter and no thousands delimiter. This environment variable is used by localeconv(3C), printf(3C), and strtod(3C).

LC_TIME
: This category specifies date and time formats. The information corresponding to this category is stored in a database specified in localedef(). The default C locale corresponds to U.S. date and time formats. This environment variable is used by many commands and functions; for example: at(1), calendar(1), date(1), strftime(3C), and getdate(3C).

MSGVERB
: Controls which standard format message components fmtmsg selects when messages are displayed to stderr; see fmtmsg(1) and fmtmsg(3C).

NETPATH
: A colon-separated list of network identifiers. A network identifier is a character string used by the Network Selection component of the system to provide application-specific default network search paths. A network identifier must consist of non-null characters and must have a length of at least 1. No maximum length is specified. Network identifiers are normally chosen by the system administrator. A network identifier is also the first field in

any /etc/netconfig file entry. NETPATH thus provides a link into the /etc/netconfig file and the information about a network contained in that network's entry. /etc/netconfig is maintained by the system administrator. The library routines described in getnetpath(3NSL) access the NETPATH environment variable.

NLSPATH

Contains a sequence of templates which catopen(3C) and gettext(3C) use when attempting to locate message catalogs. Each template consists of an optional prefix, one or more substitution fields, a filename and an optional suffix. For example:

```
NLSPATH="/system/nlslib/%N.cat"
```

defines that catopen() should look for all message catalogs in the directory /system/nlslib, where the catalog name should be constructed from the *name* parameter passed to catopen( ), %N, with the suffix .cat.

Substitution fields consist of a % symbol, followed by a single-letter keyword. The following keywords are currently defined:

%N
The value of the *name* parameter passed to catopen().

%L
The value of LANG or LC_MESSAGES.

%l
The language element from LANG or LC_MESSAGES.

%t
The territory element from LANG or LC_MESSAGES.

%c
The codeset element from LANG or LC_MESSAGES.

%%
A single % character.

An empty string is substituted if the specified value is not currently defined. The separators "_" and "." are not included in %t and %c substitutions.

Templates defined in NLSPATH are separated by colons (:). A leading colon or two adjacent colons (::) is equivalent to specifying %N. For example:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates to catopen() that it should look for the requested message catalog in *name*, *name*.cat and /nlslib/$LANG/*name*.cat. For gettext(), %N automatically maps to "messages".

If NLSPATH is unset or NULL, catopen() and gettext() call setlocale(3C), which checks LANG and the LC_* variables to locate the message catalogs.

NLSPATH will normally be set up on a system wide basis (in /etc/profile) and thus makes the location and naming conventions associated with message catalogs transparent to both programs and users.

PATH
The sequence of directory prefixes that sh(1), time(1), nice(1), nohup(1), and other utilities apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). login(1) sets PATH=/usr/bin. For more detail, see sh(1).

SEV_LEVEL
Define severity levels and associate and print strings with them in standard format error messages; see addseverity(3C), fmtmsg(1), and fmtmsg(3C).

TERM
The kind of terminal for which output is to be prepared. This information is used by commands, such as vi(1), which may exploit special capabilities of that terminal.

TZ
Timezone information. The contents of this environment variable are used by the functions ctime(3C), localtime(3C), strftime(3C), and mktime(3C) to override the default timezone. The value of TZ has one of the two formats (spaces inserted for clarity):

```
:characters
```

or

```
std offset dst offset, rule
```

If TZ is of the first format (that is, if the first character is a colon (:)), or if TZ is not of the second format, then TZ designates a path to a timezone database file relative to /usr/share/lib/zoneinfo/, ignoring a leading colon if one exists.

Otherwise, TZ is of the second form, which when expanded is as follows:

*std*offset[*dst*[*offset*][,*start*[/*time*],*end*[/*time*]]]

std and *dst*
Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (*std*) or the alternative (*dst*, such as Daylight Savings Time) timezone. Only *std* is required; if *dst* is missing, then the alternative time does not apply in this timezone. Each of these fields can occur in either of two formats, quoted or unquoted:

- In the quoted form, the first character is the less-than ('<') character and the last character is the greater-than ('>') character. All characters between these quoting characters are alphanumeric characters from the portable character set in the current locale, the plus-sign ('+') character, or the minus-sign ('-') character. The *std* and *dst* fields in this case do not include the quoting characters.
- In the unquoted form, all characters in these fields are alphabetic characters from the portable character set in the current locale.

The interpretation of these fields is unspecified if either field is less than three bytes (except for the case when *dst* is missing), more than {TZNAME_MAX} bytes, or if they contain characters other than those specified.

offset
Indicate the value one must add to the local time to arrive at Coordinated Universal Time. The offset has the form:

hh[:*mm*[:*ss*]]

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) is required and can be a single digit. The *offset* following *std* is required. If no *offset* follows *dst*, daylight savings time is assumed to be one hour ahead of standard time. One or more digits can be used. The value is always interpreted as a decimal number. The hour must be between 0 and 24, and the minutes (and seconds), if present, must be between 0 and 59. Out of range values can cause unpredictable behavior. If preceded by a "–", the timezone is east of the Prime Meridian. Otherwise, it is west of the Prime Meridian (which can be indicated by an optional preceding "+" sign).

start/*time*, *end*/*time*
Indicate when to change to and back from daylight savings time, where *start*/*time* describes when the change from standard time to daylight savings time occurs, and *end*/*time* describes when the change back occurs. Each *time* field describes when, in current local time, the change is made.

The formats of *start* and *end* are one of the following:

J*n*
The Julian day *n* ($1 \le n \le 365$). Leap days are not counted. That is, in all years, February 28 is day 59 and March 1 is day 60. It is impossible to refer to the occasional February 29.

n
The zero-based Julian day ($0 \le n \le 365$). Leap days are counted, and it is possible to refer to February 29.

M*m.n.d*
The d^{th} day, ($0 \le d \le 6$) of week *n* of month *m* of the year ($1 \le n \le 5$, $1 \le m \le 12$), where week 5 means "the last *d*-day in month *m*" which may occur in either the fourth or the fifth week). Week 1 is the first week in which the d^{th} day occurs. Day zero is Sunday.

Implementation specific defaults are used for *start* and *end* if these optional fields are not specified.

The *time* has the same format as *offset* except that no leading sign ("–" or "+") is allowed. If *time* is not specified, the default value is 02:00:00.

See Also cat(1), date(1), ed(1), fmtmsg(1), localedef(1), login(1), ls(1), mkmsgs(1), nice(1), nohup(1), sh(1), sort(1), time(1), vi(1), exec(2), addseverity(3C), catopen(3C), ctime(3C), ctype(3C), fmtmsg(3C), getdate(3C), getnetpath(3NSL), gettext(3C), gettxt(3C), localeconv(3C), mblen(3C), mktime(3C), printf(3C), setlocale(3C), strcoll(3C), strftime(3C), strtod(3C), strxfrm(3C), TIMEZONE(4), netconfig(4), passwd(4), profile(4)

Name eqnchar – special character definitions for eqn

Synopsis

```
eqn  /usr/share/lib/pub/eqnchar filename |  troff options
neqn  /usr/share/lib/pub/eqnchar filename |  troff options
```

Description The `eqnchar` command contains `nroff(1)` and `troff(1)` character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with `eqn(1)` and `neqn`(1). It contains definitions for the characters listed in the following table.

ciplus	⊕	| |	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	□
-wig	≃	*hbar*	ℏ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	✻	|<	⊀	*nomem*	∉
bigstar	✳	|>	⊁	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	⌡	*incl*	⊆
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≙	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊈
oppE	∃|	*3quarter*	¾	*!supset*	⊉
angstrom	Å	*degree*	°		

Files `/usr/share/lib/pub/eqnchar`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdoc

See Also `eqn(1)`, `nroff(1)`, `troff(1)`, `attributes(5)`

Name extendedFILE – enable extended FILE facility usage

Synopsis

```
$ ulimit -n N_file_descriptors
$ LD_PRELOAD_32=/usr/lib/extendedFILE.so.1 application [arg...]
```

Description The `extendedFILE.so.1` is not a library but an enabler of the extended FILE facility.

The extended FILE facility allows 32-bit processes to use any valid file descriptor with the standard I/O (see `stdio(3C)`) C library functions. Historically, 32-bit applications have been limited to using the first 256 numerical file descriptors for use with standard I/O streams. By using the extended FILE facility this limitation is lifted. Any valid file descriptor can be used with standard I/O. See the NOTES section of `enable_extended_FILE_stdio(3C)`.

The extended FILE facility is enabled from the shell level before an application is launched. The file descriptor limit must also be raised. The syntax for raising the file descriptor limit is

```
$ ulimit -n max_file_descriptors
$ LD_PRELOAD_32=/usr/lib/extendedFILE.so.1 application [arg...]
```

where *max_file_descriptors* is the maximum number of file descriptors desired. See `limit(1)`. The maximum value is the same as the maximum value for `open(2)`.

Environment Variables The following environment variables control the behavior of the extended FILE facility.

`_STDIO_BADFD` — This variable takes an integer representing the lowest file descriptor, which will be made unallocatable. This action provides a protection mechanism so that applications that abuse interfaces do not experience silent data corruption. The value must be between 3 and 255 inclusive.

`_STDIO_BADFD_SIGNAL` — This variable takes an integer or string representing any valid signal. See `signal.h(3HEAD)` for valid values or strings. This environment variable causes the specified signal to be sent to the application if certain exceptional cases are detected during the use of this facility. The default signal is `SIGABRT`.

Examples **EXAMPLE 1** Limit the number of file descriptors and FILE standard I/O structures.

The following example limits the number of file descriptors and FILE standard I/O structures to 1000.

```
$ ulimit -n 1000
$ LD_PRELOAD_32=/usr/lib/extendedFILE.so.1 application [arg...]
```

EXAMPLE 2 Enable the extended FILE facility.

The following example enables the extended FILE facility. See `enable_extended_FILE_stdio(3C)` for more examples.

EXAMPLE 2 Enable the extended FILE facility. *(Continued)*

```
$ ulimit -n 1000
$ _STDIO_BADFD=100 _STDIO_BADFD_SIGNAL=SIGABRT \
   LD_PRELOAD_32=/usr/lib/extendedFILE.so.1  \
   application [arg ...]
```

EXAMPLE 3 Set up the extended FILE environment and start the application.

The following shell script first sets up the proper extended FILE environment and then starts the application:

```
#!/bin/sh
if [ $# = 0 ]; then
   echo "usage: $0 application [arguments...]"
   exit 1
fi
ulimit -n 1000
# _STDIO_BADFD=196; export _STDIO_BADFD
# _STDIO_BADFD_SIGNAL=SIGABRT; export _STDIO_BADFD_SIGNAL
LD_PRELOAD_32=/usr/lib/extendedFILE.so.1; export LD_PRELOAD_32
"$@"
```

Files `/usr/lib/extendedFILE.so.1` enabling library

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl (32–bit)
Interface Stability	Stable
MT-Level	Safe

See Also `limit(1)`, `open(2)`, `enable_extended_FILE_stdio(3C)`, `fdopen(3C)`, `fopen(3C)`, `popen(3C)`, `signal.h(3HEAD)`, `stdio(3C)`, `attributes(5)`

Warnings The following displayed message

```
Application violated extended FILE safety mechanism.
Please read the man page for extendedFILE.
Aborting
```

is an indication that your application is modifying the internal file descriptor field of the `FILE` structure from standard I/O. Continued use of this extended FILE facility could harm your data. Do not use the extended FILE facility with your application.

Name extensions – localedef extensions description file

Description A localedef extensions description file or *extensions* file defines various extensions for the `localedef(1)` command.

The localedef extensions description file provides:

- EUC code set width information via the `cswidth` keyword: `cswidth bc1 : sw1,bc2 : sw2,bc3 : sw3`where `bc1`, `bc2`, and `bc3` indicate the number of bytes (byte count) per character for EUC codesets `1`, `2`, and `3`, respectively. `sw1`, `sw2`, and `sw3` indicate screen width for EUC codesets `1`, `2`, and `3`, respectively.
- Other extensions which will be documented in a future release.

See Also `locale(1)`, `localedef(1)`, `environ(5)`, `locale(5)`

Name filesystem – File system organization

Synopsis

```
/

/usr
```

Description The file system tree is organized for administrative convenience. Distinct areas within the file system tree are provided for files that are private to one machine, files that can be shared by multiple machines of a common platform, files that can be shared by all machines, and home directories. This organization allows sharable files to be stored on one machine but accessed by many machines using a remote file access mechanism such as NFS. Grouping together similar files makes the file system tree easier to upgrade and manage.

The file system tree consists of a root file system and a collection of mountable file systems. The `mount(2)` program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system or other previously mounted file systems. Two file systems, `/` (the root) and `/usr`, must be mounted and `/var` must be accessible to have a functional system. The root file system is mounted automatically by the kernel at boot time; the `/usr` file system is mounted by the system start-up script, which is run as part of the booting process. `/var` can be mounted as its own file system or be part of `/usr`, as it is by default.

Certain locations, noted below, are approved installation locations for bundled Foundation Solaris software. In some cases, the approved locations for bundled software are also approved locations for add-on system software or for applications. The following descriptions make clear where the two locations differ. For example, `/etc` is the installation location for platform-dependent configuration files that are bundled with Solaris software. The analogous location for applications is `/etc/opt/`*packagename*.

In the following descriptions, *subsystem* is a category of application or system software, such as a window system (`dt`) or a language (`java1.2`)

The following descriptions make use of the terms *platform*, *platform-dependent*, *platform-independent*, and *platform-specific*. Platform refers to a machines Instruction Set Architecture or processor type, such as is returned by `uname -i`. *Platform-dependent* refers to a file that is installed on all platforms and whose contents vary depending on the platform. Like a platform-dependent file, a *platform-independent* file is installed on all platforms. However, the contents of the latter type remains the same on all platforms. An example of a platform-dependent file is compiled, executable program. An example of a platform-independent file is a standard configuration file, such as `/etc/hosts`. Unlike a platform-dependent or a platform-independent file, the *platform-specific* file is installed only on a subset of supported platforms. Most platform-specific files are gathered under `/platform` and `/usr/platform`.

In the following file or directory descriptions, GNOME stands for GNU Network Object Model Environment. The GNOME Desktop is shipped with the Solaris operating system.

Root File System

The root file system contains files that are unique to each machine. It contains the following directories:

`/`
Root of the overall file system name space.

`/dev`
Primary location for special files. Typically, device files are built to match the kernel and hardware configuration of the machine.

`/dev/cfg`
Symbolic links to physical `ap_ids`.

`/dev/cpu`
Provides configuration and capability information about the processor type

`/dev/cua`
Device files for `uucp`.

`/dev/dsk`
Block disk devices.

`/dev/dtrace`
Pseudo-devices used by the DTrace framework.

`/dev/dtrace/provider`
Pseudo-device drivers representing instrumentation providers for the DTrace framework.

`/dev/fbs`
Frame buffer device files.

`/dev/fd`
File descriptors.

`/dev/md`
Logical volume management meta-disk devices.

`/dev/net`
Network data-link interface devices.

`/dev/printers`
USB printer device files.

`/dev/pts`
Pseudo-terminal devices.

`/dev/rdsk`
Raw disk devices.

`/dev/rmt`
Raw tape devices.

`/dev/sad`
Entry points for the STREAMS Administrative driver.

`/dev/sound`
Audio device and audio device control files.

`/dev/swap`
Default swap device.

`/dev/term`
Terminal devices.

`/devices`
Physical device files.

`/etc`
Platform-dependent administrative and configuration files and databases that are not shared among systems. `/etc` may be viewed as the directory that defines the machine's identity. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/etc/opt/`*packagename*.

`/etc/X11`
Xorg Xserver (X11) configuration files.

`/etc/acct`
Accounting system configuration information.

`/etc/apache`
Apache configuration files.

`/etc/apoc`
Files for configuring Sun Java Desktop System Configuration Manager remote access.

`/etc/bonobo-activation`
GNOME XML configuration file for identifying CORBA servers.

`/etc/cron.d`
Configuration information for `cron(1M)`.

`/etc/dat`
Contains a list of interface adapters supported by uDAPL service providers.

`/etc/default`
Defaults information for various programs.

`/etc/devices`
Contains device-related data.

`/etc/dfs`
Configuration information for shared file systems.

`/etc/dhcp`
Dynamic Host Configuration Protocol (DHCP) configuration files.

`/etc/dmi`
Solstice Enterprise Agents configuration files.

`/etc/dt`
Desktop configuration files.

`/etc/flash`
Solaris Flash Archive configuration files.

`/etc/fm`
Fault manager configuration files. For more information, see `fmd(1M)`.

`/etc/fonts`
Font configuration information.

`/etc/fs`
Binaries organized by file system types for operations required before `/usr` is mounted.

`/etc/ftpd`
`ftpd` configuration files.

`/etc/gconf`
GConf system configuration (including system defaults and system mandatory settings)

`/etc/gimp`
GNU Image Manipulation Program (GIMP) configuration files.

`/etc/gnome`
GNOME Desktop configuration files.

`/etc/gnome-vfs-2.0`
Files for customizing GNOME 2.0 desktop menus.

`/etc/gnopernicus-1.0`
Configuration files for GNOME's Gnopernicus, an Assistive Technology (AT) screen reader.

`/etc/gss`
Generic Security Service (GSS) Application Program Interface configuration files.

`/etc/gtk`
GTK+ configuration files

`/etc/gtk-2.0`
GTK+ Pixbuf loaders and Input Method modules

`/etc/imq`
Sun Java System Message Queue security configuration files.

`/etc/inet`
Configuration files for Internet services.

`/etc/init.d`
Shell scripts for transitioning between run levels.

`/etc/krb5`
Kerberos configuration files.

`/etc/lib`
Shared libraries needed during booting.

`/etc/llc2`
Logical link control (`llc2`) driver configuration files.

`/etc/lp`
Configuration information for the printer subsystem.

`/etc/lu`
Solaris Live Upgrade configuration files.

`/etc/lvm`
Solaris Logical Volume Manager configuration files.

`/etc/mail`
Mail subsystem configuration.

`/etc/nca`
Solaris Network Cache and Accelerator (NCA) configuration files.

`/etc/net`
Configuration information for transport independent network services.

`/etc/nfs`
NFS server logging configuration file.

`/etc/opt`
Configuration information for optional packages.

`/etc/pango`
Pango configuration and module information.

`/etc/patch`
Configuration files for patch management.

`/etc/ppp`
Solaris PPP configuration files.

`/etc/rc0.d`
Scripts for entering or leaving run level 0. See `init(1M)`.

`/etc/rc1.d`
Scripts for entering or leaving run level 1. See `init(1M)`.

`/etc/rc2.d`
Scripts for entering or leaving run level 2. See `init(1M)`.

`/etc/rc3.d`
Scripts for entering or leaving run level 3. See `init(1M)`.

`/etc/rcS.d`
Scripts for bringing the system up in single user mode.

`/etc/rcm`
Directory for reconfiguration manager (RCM) custom scripts.

`/etc/rpcsec`
This directory might contain an authentication configuration file.

`/etc/saf`
Service Access Facility files.

`/etc/sasl`
Simple Authentication and Security Layer (SASL) server configuration files.

`/etc/security`
Solaris-delivered security configuration files (Audit, RBAC, crypto, Trusted Extensions).

`/etc/sfw`
Platform-dependent administrative, configuration files and databases for subsystems from `/usr/sfw` that are not shared among systems.

`/etc/sfw/samba`
Samba configuration files.

`/etc/skel`
Default profile scripts for new user accounts. See `useradd(1M)`.

`/etc/sma`
Systems Management Agent (SMA) configuration files.

`/etc/snmp`
Solstice Enterprise Agents configuration files.

`/etc/sound`
Sound Events configuration files.

`/etc/ssh`
Secure Shell configuration files. See `ssh(1)`

`/etc/svc`
SMF service repository.

`/etc/sysevent`
`syseventd` configuration files.

`/etc/`*subsystem*
Platform-dependent *subsystem* configuration files that are not shared among systems. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/etc/opt/`*packagename*.

`/etc/tm`
Trademark files; contents displayed at boot time.

`/etc/usb`
USB configuration information.

`/etc/uucp`
UUCP configuration information. See `uucp(1C)`.

`/etc/xml`
Extensible Markup Language (XML) catalog.

`/etc/zones`
Solaris Zones configuration files.

`/export`
Default root of the shared file system tree.

`/home`
Default root of a subtree for user directories.

`/kernel`
Subtree of platform-dependent loadable kernel modules required as part of the boot process. It includes the generic part of the core kernel that is platform-independent, `/kernel/genunix`. See `kernel(1M)` An approved installation location for bundled Solaris software and for add-on system software.

`/kernel/drv`
32-bit x86 device drivers.

`/kernel/drv/sparcv9`
64-bit SPARC device drivers.

`/kernel/drv/amd64`
64-bit device drivers for 64–bit x86 platforms.

`/kernel/dtrace`
Kernel modules representing components in the DTrace framework.

`/kernel/genunix`
Platform-independent kernel.

`/kernel/amd64/genunix`
64–bit, platform-independent kernel.

`/kernel/`*subsystem*`/amd64`
64-bit x86 platform-dependent modules required for boot. An approved installation location for bundled Solaris software and for add-on system software.

`/kernel/`*subsystem*`/sparcv9`
64-bit SPARC platform-dependent modules required for boot. An approved installation location for bundled Solaris software and for add-on system software.

`/lib/svc/manifest`
SMF method scripts. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/packagename/lib/svc/manifest`.

`/mnt`
Default temporary mount point for file systems. This is an empty directory on which file systems can be temporarily mounted.

`/net`
Temporary mount point for file systems that are mounted by the automounter.

`/opt`
Root of a subtree for add-on application packages.

`/platform`
Subtree of platform-specific objects which need to reside on the root filesystem. It contains a series of directories, one per supported platform. The semantics of the series of directories is equivalent to `/` (root).

`/platform/‘uname -i‘/kernel`
Platform-specific modules required for boot. These modules have semantics equivalent to `/kernel`. It includes the file `unix`, the core kernel. See `kernel(1M)`. An approved installation location for bundled Solaris software and for add-on system software.

`/platform/‘uname -m‘/kernel`
Hardware class-specific modules required for boot. An approved installation location for bundled Solaris software and for add-on system software.

`/platform/‘uname -i‘/kernel/`*subsystem*`/amd64`
x86 64-bit, platform-dependent modules required for boot. This is an approved installation location for bundled Solaris software.

`/platform/‘uname -i‘/kernel/`*subsystem*`/sparcv9`
SPARC 64-bit platform-specific modules required for boot. An approved installation location for bundled Solaris software.

`/platform/‘uname -i‘/kernel/sparcv9/unix`
64-bit platform-dependent kernel.

`/platform/‘uname -i‘/kernel/unix`
32-bit platform-dependent kernel on i86 and a symlink to `sparcv9/unix` on SPARC.

`/platform/‘uname -i‘/lib`
Platform-specific shared objects required for boot. Semantics are equivalent to `/lib`. An approved installation location for bundled Solaris software and for add-on system software.

`/platform/‘uname -i‘/sbin`
Platform-specific administrative utilities required for boot. Semantics are equivalent to `/sbin`. An approved installation location for bundled Solaris software and for add-on system software.

`/proc`
Root of a subtree for the process file system.

`/sbin`
Essential executables used in the booting process and in manual system recovery. The full complement of utilities is available only after `/usr` is mounted. `/sbin` is an approved installation location for bundled Solaris software.

`/system`
Mount point for the contract (CTFS) and object (OBJFS) file systems.

`/tmp`
Temporary files; cleared during the boot operation.

`/usr`
Mount point for the `/usr` file system. See description of `/usr` file system, below.

`/var`
Root of a subtree for varying files. Varying files are files that are unique to a machine but that can grow to an arbitrary (that is, variable) size. An example is a log file. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/var/opt/`*packagename*.

`/var/adm`
System logging and accounting files.

`/var/apache`
Scripts, icons, logs, and cache pages for Apache web server.

`/var/appserver`
Sun Java System Application Server administrative domain files.

`/var/audit`
Default location for Solaris Audit log files.

`/var/cores`
Directory provided for global core files storage.

`/var/crash`
Default depository for kernel crash dumps.

`/var/cron`
Log files for `cron(1M)`.

`/var/dmi`
Solstice Enterprise Agents (SEA) Desktop Management Interface (DMI) run-time components.

`/var/dt`
`dtlogin` configuration files.

`/var/fm`
Fault manager state files. For more information, see `fmd(1M)`.

`/var/imq`
Message queue broker instance configuration file.

`/var/ftp`
FTP server directory.

`/var/inet`
IPv6 router state files.

`/var/krb5`
Database and log files for Kerberos.

`/var/ld`
Configuration files for runtime linker.

`/var/ldap`
LDAP client configuration files.

`/var/lib`
Directory for variable state information.

`/var/log`
System log files.

`/var/lp`
Line printer subsystem logging information.

`/var/mail`
Directory where users' mail is kept.

`/var/mysql`
Dynamic database directory for MySQL Database Management System.

`/var/news`
Community service messages. This is not the same as USENET-style news.

`/var/nfs`
NFS server log files.

`/var/ntp`
Network Time Protocol (NTP) server state directory.

`/var/opt`
Root of a subtree for varying files associated with optional software packages. An approved installation location for add-on system software and applications.

`/var/patchsrv`
Patch management log files.

`/var/preserve`
Backup files for `vi(1)` and `ex(1)`.

`/var/run`
Temporary files which are not needed across reboots. Only root may modify the contents of this directory.

`/var/sadm`
Databases maintained by the software package management utilities.

`/var/sadm/system/logs`
Status log files produced by software management functions and/or applications. For example, log files produced for product installation. An approved installation location for bundled Solaris software and for add-on system software and applications.

`/var/saf`
Service access facility logging and accounting files.

`/var/samba`
Log and lock files for Samba.

`/var/sma_snmp`
Systems Management Agent (SMA) security and MIB component information.

`/var/snmp`
SNMP status and configuration information.

`/var/spool`
Contains directories for files used in printer spooling, mail delivery, `cron(1M)`, `at(1)`, and so forth.

`/var/spool/clientmqueue`
`sendmail(1M)` client files.

`/var/spool/cron`
`cron(1M)` and `at(1)` spooling files.

`/var/spool/locks`
Spooling lock files.

`/var/spool/lp`
Line printer spool files. See `lp(1)`.

`/var/spool/mqueue`
Mail queued for delivery.

`/var/spool/pkg`
Spooled packages.

`/var/spool/print`
LP print service client-side request staging area.

`/var/spool/samba`
Samba print queue.

`/var/spool/uucp`
Queued `uucp(1C)` jobs.

`/var/spool/uucppublic`
Files deposited by `uucp(1C)`.

`/var/statmon`
Network status monitor files.

`/var/svc/log`
SMF log files.

`/var/svc/manifest`
SMF service manifests. An approved installation location for bundled, add-on system software and applications.

`/var/svc/manifest/site`
Site-local SMF service manifests.

`/var/tmp`
Files that vary in size or presence during normal system operations. This directory is *not* cleared during the boot operation. An approved installation location for bundled Solaris software and for add-on system software and applications.

It is possible to change the default behavior for `/var/tmp` to clear all of the files except editor temporary files by setting the `clean_vartmp` property value of the `rmtmpfiles` service. This is done with the following commands:

```
# svccfg -s svc:/system/rmtmpfiles setprop \
      options/clean_vartmp = "true"
# svcadm refresh svc:/system/rmtmpfiles:default
```

The `solaris.smf.value.rmtmpfiles` authorization is required to modify this property.

`/var/uucp`
`uucp(1C)` log and status files.

`/var/yp`
Databases needed for backwards compatibility with NIS and `ypbind(1M)`.

`/usr` File System

Because it is desirable to keep the root file system small and not volatile, on disk-based systems larger file systems are often mounted on `/home`, `/opt`, `/usr`, and `/var`.

The file system mounted on `/usr` contains platform-dependent and platform-independent sharable files. The subtree rooted at `/usr/share` contains platform-independent sharable files; the rest of the `/usr` tree contains platform-dependent files. By mounting a common remote

file system, a group of machines with a common platform may share a single `/usr` file system. A single `/usr/share` file system can be shared by machines of any platform. A machine acting as a file server can share many different `/usr` file systems to support several different architectures and operating system releases. Clients usually mount `/usr` read-only so that they do not accidentally change any shared files.

The `/usr` file system contains the following subdirectories:

`/usr/4lib`	`a.out` libraries for the Binary Compatibility Package.
`/usr/5bin`	Symbolic link to the `/usr/bin` directory.
`/usr/SUNWale`	Configuration files for Asian Lanuguage Environment (ALE).
`/usr/X`	Symbolic link to the `/usr/openwin` directory.
`/usr/X11`	Xorg Xserver (X11) executables and documentation.
`/usr/adm`	Symbolic link to the `/var/adm` directory.
`/usr/apache`	Apache executables, loadable modules, and documentation.
`/usr/appserver`	Sun Java System Application Server software.
`/usr/benchmarks`	Directory for benchmarks.
`/usr/bin`	Platform-dependent, user-invoked executables. These are commands users expect to be run as part of their normal `$PATH`. For executables that are different on a 64–bit system than on a 32–bit system, a wrapper that selects the appropriate executable is placed here. See `isaexec(3C)`. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/bin`.

`/usr/bin/amd64`	x86 64–bit, platform-dependent, user-invoked executables. This directory should not be part of a user's `$PATH`. A wrapper in `/usr/bin` should invoke the executable in this directory. See `isaexec`(3C). An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/bin/amd64`.
`/usr/bin/sparcv9`	SPARC platform-dependent, user-invoked executables. This directory should not be part of a user's `$PATH`. A wrapper in `/usr/bin` should invoke the executable in this directory. See `isaexec`(3C). An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/bin/sparcv9`.
`/usr/bin/amd64`	x86 platform-dependent, user-invoked executables. This directory should not be part of a user's `$PATH`. A wrapper in `/usr/bin` should invoke the executable in this directory. See `isaexec`(3C). An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/bin/amd64`.
`/usr/bin/`*subsystem*	Platform-dependent user-invoked executables that are associated with *subsystem*. These are commands users expect to be run as part of their normal `$PATH`. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/bin`.

/usr/*subsystem*/bin	Platform-dependent user-invoked executables that are associated with *subsystem*. These are commands users expect to be run as part of their normal $PATH. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is /opt/*packagename*/bin.
/usr/*subsystem*/bin/amd64	x86 64–bit, platform-dependent, user-invoked executables. This directory should not be part of a user's $PATH. A wrapper in /usr/bin should invoke the executable in this directory. See isaexec(3C). An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is /opt/*packagename*/bin/amd64.
/usr/*subsystem*/bin/sparcv9	SPARC 64–bit, platform-dependent, user-invoked executables. This directory should not be part of a user's $PATH. A wrapper in /usr/bin should invoke the executable in this directory. See isaexec(3C). An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is /opt/*packagename*/bin/sparcv9.
/usr/ccs	C compilation system.
/usr/ccs/bin	C compilation commands and system utilities.
/usr/ccs/lib	Symbolic link to /usr/lib.
/usr/demo	Demo programs and data.
/usr/dict	Symbolic link to the /usr/share/lib/dict directory, which contains the dictionary file used by the UNIX spell program.

`/usr/dt`	root of a subtree for CDE software.
`/usr/dt/bin`	Primary location for CDE system utilities.
`/usr/dt/include`	Header files for CDE software.
`/usr/dt/lib`	Libraries for CDE software.
`/usr/dt/share/man`	On-line reference manual pages for CDE software.
`/usr/games`	An empty directory, a remnant of the SunOS 4.0/4.1 software.
`/usr/gnome`	This is an obsolete directory where 3rd party programs can install their applications and `pixmap` files. It is supported for backwards compatibility.
`/usr/include`	Include headers (for C programs).
`/usr/j2se`	Java 2 SDK executables, loadable modules, and documentation.
`/usr/java*`	Directories containing Java programs and libraries.
`/usr/jdk*`	Java Platform virtual machine and core class libraries.
`/usr/kernel`	Subtree of platform-dependent loadable kernel modules, not needed in the root filesystem. An approved installation location for bundled Solaris software.
`/usr/kvm`	A mount point, retained for backward compatibility, that formerly contained platform-specific binaries and libraries.
`/usr/lib`	Platform-dependent libraries, various databases, commands and daemons not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system

	software or for applications is `/opt/`*packagename*`/lib`.
`/usr/lib/32`	Symbolic link to `/usr/lib`.
`/usr/lib/64`	Symbolic link to the most portable 64-bit Solaris interfaces, on both SPARC and x86 platforms.
`/usr/lib/acct`	Accounting scripts and binaries. See `acct(1M)`.
`/usr/lib/adb`	`adb` accounting scripts.
`/usr/lib/amd64`	Platform-dependent libraries, various databases, commands and daemons not invoked directly by a human user on 64–bit x86. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/amd64`.
`/usr/lib/autofs`	Contains the `automountd` executable.
`/usr/lib/cfgadm`	Contains `cfgadm` hardware–specific driver plugins.
`/usr/lib/class`	Scheduling-class-specific directories containing executables for `priocntl(1)` and `dispadmin(1M)`.
`/usr/lib/crypto`	Contains the kernel-level cryptographic framework daemon (`kcfd`).
`/usr/lib/devfsadm`	Contains `devfsadm`, the daemon version of `devfsadm`.
`/usr/lib/dict`	Database files for `spell(1)`.
`/usr/lib/dns`	Contains DNS resolver libraries.
`/usr/lib/dtrace`	Contains `dtrace` D source files.
`/usr/lib/flash`	Contains Solaris flash archive deployment scripts.
`/usr/lib/fm`	Contains `fmd`, the fault manager daemon and the fault manager library.

`/usr/lib/font`	`troff(1)` font description files.
`/usr/lib/fs`	File system type dependent modules; generally not intended to be invoked directly by the user.
`/usr/lib/gss`	Secure services-related libraries.
`/usr/lib/iconv`	Conversion tables for `iconv(1)`.
`/usr/lib/inet`	Contains many network-related daemons and libraries.
`/usr/lib/ipf`	Contains `IPFILTER.LICENCE` and `ipftest`.
`/usr/lib/ipqosconf`	IPQoS configuration utility.
`/usr/lib/krb5`	Contains the Kerberos database propagation program and libraries.
`/usr/lib/ld`	Contains the map files for the `ld` link editor.
`/usr/lib/ldap`	Contains LDAP client configuration utilities.
`/usr/lib/libp`	Profiled libraries.
`/usr/lib/llc2`	Contains logical link control (`llc2`) driver configuration files.
`/usr/lib/locale`	Localization databases.
`/usr/lib/lp`	Line printer subsystem databases and back-end executables.
`/usr/lib/lu`	Live Upgrade utilities.
`/usr/lib/netsvc`	Internet network services.
`/usr/lib/nfs`	Auxiliary NFS-related programs and daemons.
`/usr/lib/picl`	Platform Information and Control Library.
`/usr/lib/pool`	Contains the automated resource pools partitioning daemon (`poold`) and associated libraries.
`/usr/lib/power`	Power management daemon, `powerd`.

`/usr/lib/print`	Contains `lp` conversion scripts and the `in.lpd` daemon.
`/usr/lib/rcap`	Resource cap enforcement daemon, `rcapd`.
`/usr/lib/rcm`	Contains the Reconfiguration and Coordination Manager daemon (`rcm_daemon`) and RCM scripts.
`/usr/lib/refer`	Auxiliary programs for `refer(1)`.
`/usr/lib/rmmount`	Removable media mounter shared objects.
`/usr/lib/sa`	Scripts and commands for the system activity report package. See `sar(1)`.
`/usr/lib/saf`	Auxiliary programs and daemons related to the service access facility.
`/usr/lib/sasl`	Simple Authentication and Security Layer (SASL) plug-in modules.
`/usr/lib/secure`	Default trusted libraries.
`/usr/lib/security`	Solaris security plug-in modules.
`/usr/lib/smedia`	Removable media device server daemon, `rpc.smserverd`.
`/usr/lib/sparcv9`	SPARC 64-bit, platform-dependent libraries, various databases, commands and daemons not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/sparcv9`.
`/usr/lib/spell`	Auxiliary programs and databases for `spell(1)`. This directory is only present when the Binary Compatibility Package is installed.
`/usr/lib/ssh`	Contains the Secure Shell daemon (`sshd`) and supporting programs.

`/usr/lib/`*subsystem*	Platform-dependent libraries, various databases, commands and daemons that are associated with *subsystem* and that are not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib`.
`/usr/lib/`*subsystem*`/amd64`	x86 64–bit, platform-dependent libraries, various databases, commands and daemons that are associated with *subsystem* and that are not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/amd64`.
`/usr/lib/`*subsystem*`/sparcv9`	SPARC 64-bit, platform-dependent libraries, various databases, commands and daemons that are associated with *subsystem* and that are not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/sparcv9`.
`/usr/`*subsystem*`/lib`	Platform-dependent libraries, various databases, commands and daemons not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib`.
`/usr/`*subsystem*`/lib/amd64`	x86 64–bit, platform-dependent libraries, various databases, commands and daemons that are associated with *subsystem* and that are

	not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/amd64`.
`/usr/`*subsystem*`/lib/sparcv9`	SPARC 64-bit, platform-dependent libraries, various databases, commands and daemons that are associated with *subsystem* and that are not invoked directly by a human user. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/lib/sparcv9`.
`/usr/lib/sysevent`	Contains the system event notification daemon (`syseventd`) and the `syseventd` loadable module (SLM) repository.
`/usr/lib/uucp`	Auxiliary programs and daemons for `uucp`(1C).
`/usr/lib/webconsole`	Sun Java web console programs and scripts.
`/usr/lib/zones`	Zone administration daemon (`zoneamd`).
`/usr/local`	Not part of the SVR4-based Solaris distribution. The `/usr` directory is exclusively for software bundled with the Solaris operating system. If needed for storing machine-local add-on software, create the directory `/opt/local` and make `/usr/local` a symbolic link to `/opt/local`. The `/opt` directory or filesystem is for storing add-on software to the system.
`/usr/mail`	Symbolic link to the `/var/mail` directory.

`/usr/man`	Symbolic link to the `/usr/share/man` directory.
`/usr/net/servers`	Entry points for foreign name service requests relayed using the network listener. See `listen(1M)`.
`/usr/news`	Symbolic link to the `/var/news` directory.
`/usr/old`	Programs that are being phased out.
`/usr/openwin`	Installation or mount point for the OpenWindows software.
`/usr/perl5`	Perl 5 programs and documentation
`/usr/platform`	Subtree of platform-specific objects which does not need to reside on the root filesystem. It contains a series of directories, one per supported platform. The semantics of the series of directories is equivalent to `/platform`, except for subdirectories which do not provide utility under one or the other (for example, `/platform/include` is not needed).
`/usr/platform/‘uname -i‘/include`	Symbolic link to `/../‘uname -i‘/include`. Platform-specific system (`sys`, `vm`) header files with semantics equivalent to `/usr/include`. An approved installation location for bundled Solaris software and for add-on system software.
`/usr/platform/‘uname -i‘/lib`	Platform-specific shared objects with semantics equivalent to `/usr/lib`. An approved installation location for bundled Solaris software and for add-on system software.
`/usr/platform/‘uname -i‘/lib/`*subsystem*`/amd64`	x86 64–bit, platform-specific daemon and shared objects. An approved installation location for bundled Solaris software and for add-on system software.

`/usr/platform/‘uname -i‘/sbin`	Platform-specific system administration utilities with semantics equivalent to `/usr/sbin`. An approved installation location for bundled Solaris software and for add-on system software.
`/usr/preserve`	Symbolic link to the `/var/preserve` directory.
`/usr/proc`	Directory for the `proc` tools.
`/usr/pub`	Symbolic link to `/share/lib/pub`, which contains files for online man page and character processing.
`/usr/sadm`	System administration files and directories.
`/usr/sadm/bin`	Binaries for the Form and Menu Language Interpreter (FMLI) scripts. See `fmli(1)`.
`/usr/sadm/install`	Executables and scripts for package management.
`/usr/sbin`	Platform-dependent executables for system administration, expected to be run only by system administrators. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/sbin`.
`/usr/sbin/install.d`	Custom Jumpstart scripts and executables.
`/usr/sbin/sparc7` and `sparc9`	32–bit and 64–bit versions of commands.
`/usr/sbin/amd64`	64–bit x86versions of commands.
`/usr/sbin/`*subsystem*	Platform-dependent executables for system administration, expected to be run only by system administrators, and associated with *subsystem*. An approved installation location for bundled Solaris software. The

	analogous location for add-on system software or for applications is `/opt/`*packagename*`/sbin`.
`/usr/`*subsystem*`/sbin`	Platform-dependent executables for system administration, expected to be run only by system administrators, and associated with *subsystem*. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/sbin`.
`/usr/sfw`	GNU and open source executables, libraries, and documentation.
`/usr/share`	Platform-independent sharable files. An approved installation location for bundled Solaris software.
`/usr/share/aclocal`	Open source `m4` files.
`/usr/share/applications`	Open source desktop applications files.
`/usr/share/audio`	Sample audio files.
`/usr/share/glib-2.0`	Makefile for `glib`.
`/usr/share/gnome`	GNOME desktop and application data.
`/usr/share/gtk-2.0`	GTK+ application data and demos
`/usr/share/gtk-doc`	API documentation for libraries which use gtk-doc documentation format, which mostly includes desktop interfaces.
`/usr/share/icons`	Sun Java Desktop icons.
`/usr/share/idl`	Open source Interface Definition Language (IDL) files.
`/usr/share/intltool`	XML translation tools.
`/usr/share/ipfilter`	Open source IP Filter sample files.
`/usr/share/javadoc`	Help files for Message Queue broker and Smart Card applications.

`/usr/share/lib`	Platform-independent sharable databases. An approved installation location for bundled Solaris software.
`/usr/share/lib/dict`	Contains word list for `spell(1)`.
`/usr/share/lib/keytables`	Keyboard layout description tables.
`/usr/share/lib/mailx`	Help files for `mailx(1)`.
`/usr/share/lib/nterm`	`nroff(1)` terminal tables.
`/usr/share/lib/pub`	Character set data files.
`/usr/share/lib/tabset`	Tab setting escape sequences.
`/usr/share/lib/terminfo`	Terminal description files for `terminfo(4)`.
`/usr/share/lib/tmac`	Macro packages and related files for text processing tools, for example, `nroff(1)` and `troff(1)`.
`/usr/share/lib/zoneinfo`	Time zone information.
`/usr/share/man`	Platform-independent sharable manual pages. An approved installation location for bundled Solaris software. The analogous location for add-on system software or for applications is `/opt/`*packagename*`/man`.
`/usr/share/omf`	GNOME Scrollkeeper database files.
`/usr/share/pixmaps`	Sun Java graphics.
`/usr/share/scrollkeeper`	GNOME Scrollkeeper templates and `xslt` files.
`/usr/share/sgml`	Open source SGML files.
`/usr/share/sounds`	Sound files.
`/usr/share/src`	Source code for kernel, utilities, and libraries.
`/usr/share/themes`	GNOME 2.0 Desktop themes.
`/usr/share/webconsole`	Sun Web Console status files.
`/usr/share/xml`	GNOME Scrollkeeper DTD files.

`/usr/snadm`	Files related to system and network administration.
`/usr/spool`	Symbolic link to the `/var/spool` directory.
`/usr/src`	Symbolic link to the `/usr/share/src` directory.
`/usr/tmp`	Symbolic link to the `/var/tmp` directory.
`/usr/ucb`	Berkeley compatibility package binaries.
`/usr/ucbinclude`	Berkeley compatibility package headers.
`/usr/ucblib`	Berkeley compatibility package libraries.
`/usr/xpg4`	Directory for POSIX-compliant utilities.
`/usr/xpg6`	Directory for newer versions of POSIX-compliant utilities.

See Also `at(1)`, `ex(1)`, `fmli(1)`, `iconv(1)`, `lp(1)`, `isainfo(1)`, `mail(1)`, `mailx(1)`, `nroff(1)`, `priocntl(1)`, `refer(1)`, `sar(1)`, `sh(1)`, `spell(1)`, `svcs(1)`, `troff(1)`, `uname(1)`, `uucp(1C)`, `vi(1)`, `acct(1M)`, `cron(1M)`, `dispadmin(1M)`, `dladm(1M)`, `fmd(1M)`, `fsck(1M)`, `init(1M)`, `kernel(1M)`, `mknod(1M)`, `mount(1M)`, `svcadm(1M)`, `svccfg(1M)`, `useradd(1M)`, `ypbind(1M)`, `mount(2)`, `Intro(4)`, `terminfo(4)`

Name fnmatch – file name pattern matching

Description The pattern matching notation described below is used to specify patterns for matching strings in the shell. Historically, pattern matching notation is related to, but slightly different from, the regular expression notation. For this reason, the description of the rules for this pattern matching notation is based on the description of regular expression notation described on the `regex(5)` manual page.

Patterns Matching a Single Character

The following *patterns matching a single character* match a single character: *ordinary characters*, *special pattern characters* and *pattern bracket expressions*. The pattern bracket expression will also match a single collating element.

An ordinary character is a pattern that matches itself. It can be any character in the supported character set except for `NUL`, those special shell characters that require quoting, and the following three special pattern characters. Matching is based on the bit pattern used for encoding the character, not on the graphic representation of the character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern will match the character itself. The shell special characters always require quoting.

When unquoted and outside a bracket expression, the following three characters will have special meaning in the specification of patterns:

`?` A question-mark is a pattern that will match any character.

`*` An asterisk is a pattern that will match multiple characters, as described in `Patterns Matching Multiple Characters`, below.

`[` The open bracket will introduce a pattern bracket expression.

The description of basic regular expression bracket expressions on the `regex(5)` manual page also applies to the pattern bracket expression, except that the exclamation-mark character `( ! )` replaces the circumflex character (`^`) in its role in a *non-matching list* in the regular expression notation. A bracket expression starting with an unquoted circumflex character produces unspecified results.

The restriction on a circumflex in a bracket expression is to allow implementations that support pattern matching using the circumflex as the negation character in addition to the exclamation-mark. A portable application must use something like `[\^!]` to match either character.

When pattern matching is used where shell quote removal is not performed (such as in the argument to the `find –name` primary when `find` is being called using one of the `exec` functions, or in the *pattern* argument to the `fnmatch(3C)` function, special characters can be escaped to remove their special meaning by preceding them with a backslash character. This escaping backslash will be discarded. The sequence `\\` represents one literal backslash. All of the requirements and effects of quoting on ordinary, shell special and special pattern characters will apply to escaping in this context.

Both quoting and escaping are described here because pattern matching must work in three separate circumstances:

- Calling directly upon the shell, such as in pathname expansion or in a `case` statement. All of the following will match the string or file `abc`:

`abc`	`"abc"`	`a"b"c`	`a\bc`	`a[b]c`
`a["b"]c`	`a[\b]c`	`a["\b"]c`	`a?c`	`a*c`

  The following will not:

`"a?c"`	`a\*c`	`a\[b]c`

- Calling a utility or function without going through a shell, as described for `find(1)` and the function `fnmatch(3C)`
- Calling utilities such as `find`, `cpio`, `tar` or `pax` through the shell command line. In this case, shell quote removal is performed before the utility sees the argument. For example, in:

  find /bin -name e\c[\h]o -print

  after quote removal, the backslashes are presented to `find` and it treats them as escape characters. Both precede ordinary characters, so the `c` and `h` represent themselves and `echo` would be found on many historical systems (that have it in `/bin`). To find a file name that contained shell special characters or pattern characters, both quoting and escaping are required, such as:

  ```
  pax -r . . . "*a\ ( \?"
  ```

  to extract a filename ending with `a(?`.

Conforming applications are required to quote or escape the shell special characters (sometimes called metacharacters). If used without this protection, syntax errors can result or implementation extensions can be triggered. For example, the KornShell supports a series of extensions based on parentheses in patterns; see `ksh(1)`

Patterns Matching Multiple Characters

The following rules are used to construct *patterns matching multiple characters* from *patterns matching a single character*:

- The asterisk (*) is a pattern that will match any string, including the null string.
- The concatenation of *patterns matching a single character* is a valid pattern that will match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.

- The concatenation of one or more *patterns matching a single character* with one or more asterisks is a valid pattern. In such patterns, each asterisk will match a string of zero or more characters, matching the greatest possible number of characters that still allows the remainder of the pattern to match the string.

Since each asterisk matches zero or more occurrences, the patterns `a*b` and `a**b` have identical functionality.

Examples:

`a[bc]` matches the strings `ab` and `ac`.

`a*d` matches the strings `ad`, `abd` and `abcd`, but not the string `abc`.

`a*d*` matches the strings `ad`, `abcd`, `abcdef`, `aaaad` and `adddd`.

`*a*d` matches the strings `ad`, `abcd`, `efabcd`, `aaaad` and `adddd`.

Patterns Used for Filename Expansion

The rules described so far in `Patterns Matching Multiple Characters` and `Patterns Matching a Single Character` are qualified by the following rules that apply when pattern matching notation is used for filename expansion.

1. The slash character in a pathname must be explicitly matched by using one or more slashes in the pattern; it cannot be matched by the asterisk or question-mark special characters or by a bracket expression. Slashes in the pattern are identified before bracket expressions; thus, a slash cannot be included in a pattern bracket expression used for filename expansion. For example, the pattern `a[b/c]d` will not match such pathnames as `abd` or `a/d`. It will only match a pathname of literally `a[b/c]d`.
2. If a filename begins with a period (.), the period must be explicitly matched by using a period as the first character of the pattern or immediately following a slash character. The leading period will not be matched by:

   • the asterisk or question-mark special characters

   • a bracket expression containing a non-matching list, such as:

   `[!a]`

   a range expression, such as:

   `[%–0]`

   or a character class expression, such as:

   `[[:punct:]]`

   It is unspecified whether an explicit period in a bracket expression matching list, such as:

   `[.abc]`

   can match a leading period in a filename.

3. Specified patterns are matched against existing filenames and pathnames, as appropriate. Each component that contains a pattern character requires read permission in the directory containing that component. Any component, except the last, that does not contain a pattern character requires search permission. For example, given the pattern:

   `/foo/bar/x*/bam`

   search permission is needed for directories `/` and `foo`, search and read permissions are needed for directory `bar`, and search permission is needed for each `x*` directory.

   If the pattern matches any existing filenames or pathnames, the pattern will be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale. If the pattern contains an invalid bracket expression or does not match any existing filenames or pathnames, the pattern string is left unchanged.

See Also `find(1)`, `ksh(1)`, `fnmatch(3C)`, `regex(5)`

Name formats – file format notation

Description Utility descriptions use a syntax to describe the data organization within files—stdin, stdout, stderr, input files, and output files—when that organization is not otherwise obvious. The syntax is similar to that used by the `printf(3C)` function. When used for stdin or input file descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the `scanf(3C)` function to read the input file.

Format The description of an individual record is as follows:

"<format>", [<*arg1*>, <*arg2*>, . . ., <*argn*>]

The `format` is a character string that contains three types of objects defined below:

characters	Characters that are not *escape sequences* or *conversion specifications*, as described below, are copied to the output.
escape sequences	Represent non-graphic characters.
conversion specifications	Specifies the output format of each argument. (See below.)

The following characters have the following special meaning in the format string:

„ (An empty character position.) One or more blank characters.

/\ Exactly one space character.

The notation for spaces allows some flexibility for application output. Note that an empty character position in `format` represents one or more blank characters on the output (not *white space*, which can include newline characters). Therefore, another utility that reads that output as its input must be prepared to parse the data using `scanf(3C)`, `awk(1)`, and so forth. The character is used when exactly one space character is output.

Escape Sequences The following table lists escape sequences and associated actions on display devices capable of the action.

Sequence	Character	Terminal Action
\\	backslash	None.
\a	alert	Attempts to alert the user through audible or visible notification.
\b	backspace	Moves the printing position to one column before the current position, unless the current position is the start of a line.
\f	form-feed	Moves the printing position to the initial printing position of the next logical page.

Sequence	Character	Terminal Action
\n	newline	Moves the printing position to the start of the next line.
\r	carriage-return	Moves the printing position to the start of the current line.
\t	tab	Moves the printing position to the next tab position on the current line. If there are no more tab positions left on the line, the behavior is undefined.
\v	vertical-tab	Moves the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

Conversion Specifications

Each conversion specification is introduced by the percent-sign character (%). After the character %, the following appear in sequence:

flags — Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

field width — An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it is padded on the left (or right, if the left-adjustment flag (–), described below, has been given to the field width).

precision — Gives the minimum number of digits to appear for the d, o, i, u, x or X conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the e and f conversions, the maximum number of significant digits for the g conversion; or the maximum number of bytes to be written from a string in s conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero.

conversion characters — A conversion character (see below) that indicates the type of conversion to be applied.

flags

The *flags* and their meanings are:

– The result of the conversion is left-justified within the field.

+ The result of a signed conversion always begins with a sign (+ or –).

<*space*> If the first character of a signed conversion is not a sign, a space character is prefixed to the result. This means that if the space character and + flags both appear, the space character flag is ignored.

The value is to be converted to an alternative form. For c, d, i, u, and s conversions, the behaviour is undefined. For o conversion, it increases the

precision to force the first digit of the result to be a zero. For x or X conversion, a non-zero result has 0x or 0X prefixed to it, respectively. For e, E, f, g, and G conversions, the result always contains a radix character, even if no digits follow the radix character. For g and G conversions, trailing zeros are not removed from the result as they usually are.

0 For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and – flags both appear, the 0 flag is ignored. For d, i, o, u, x and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behaviour is undefined.

Conversion Characters

Each conversion character results in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are ignored.

The *conversion characters* and their meanings are:

d,i,o,u,x,X The integer argument is written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers convert to signed decimal in the style `[−]`*dddd*. The x conversion uses the numbers and letters 0123456789abcdef and the X conversion uses the numbers and letters 0123456789ABCDEF. The *precision* component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters. If both the field width and precision are omitted, the implementation may precede, follow or precede and follow numeric arguments of types d, i and u with blank characters; arguments of type o (octal) may be preceded with leading zeros.

The treatment of integers and spaces is different from the `printf(3C)` function in that they can be surrounded with blank characters. This was done so that, given a format such as:

`"%d\n",`<*foo*>

the implementation could use a `printf()` call such as:

`printf("%6d\n",` *foo*`);`

and still conform. This notation is thus somewhat like `scanf()` in addition to `printf( ).`

f The floating point number argument is written in decimal notation in the style `[−]`*ddd.ddd*, where the number of digits after the radix character (shown here as a decimal point) is equal to the *precision* specification. The `LC_NUMERIC` locale category determines the radix character to use in this format. If the

precision is omitted from the argument, six digits are written after the radix character; if the *precision* is explicitly 0, no radix character appears.

e,E The floating point number argument is written in the style `[−]`*d.ddd*e±`dd` (the symbol ± indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The `LC_NUMERIC` locale category determines the radix character to use in this format. When the precision is missing, six digits are written after the radix character; if the precision is 0, no radix character appears. The E conversion character produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits are written as necessary.

g,G The floating point number argument is written in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted: style g is used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.

c The integer argument is converted to an `unsigned char` and the resulting byte is written.

s The argument is taken to be a string and bytes from the string are written until the end of the string or the number of bytes indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it is taken to be infinite, so all bytes up to the end of the string are written.

% Write a % character; no argument is converted.

In no case does a non-existent or insufficient *field width* cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term *field width* should not be confused with the term *precision* used in the description of %s.

One difference from the C function `printf()` is that the l and h conversion characters are not used. There is no differentiation between decimal values for type `int`, type `long`, or type `short`. The specifications %d or %i should be interpreted as an arbitrary length sequence of digits. Also, no distinction is made between single precision and double precision numbers (`float` or `double` in C). These are simply referred to as floating point numbers.

Many of the output descriptions use the term `line`, such as:

"`%s`", <*input line*>

Since the definition of `line` includes the trailing newline character already, there is no need to include a `\n` in the format; a double newline character would otherwise result.

Examples

EXAMPLE 1 To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where <*weekday*> and <*month*> are strings:

"`%s,/\%s/\%d,/\%d:%.2d\n`",<*weekday*>,<*month*>,<*day*>,<*hour*>,<*min*>

EXAMPLE 2 To show pi written to 5 decimal places:

"`pi/\=/\%.5f\n`",<*value of pi*>

EXAMPLE 3 To show an input file format consisting of five colon-separated fields:

"`%s:%s:%s:%s:%s\n`",<*arg1*>,<*arg2*>,<*arg3*>,<*arg4*>,<*arg5*>

See Also

`awk`(1), `printf`(1), `printf`(3C), `scanf`(3C)

Name fsattr – extended file attributes

Description Attributes are logically supported as files within the file system. The file system is therefore augmented with an orthogonal name space of file attributes. Any file (including attribute files) can have an arbitrarily deep attribute tree associated with it. Attribute values are accessed by file descriptors obtained through a special attribute interface. This logical view of "attributes as files" allows the leveraging of existing file system interface functionality to support the construction, deletion, and manipulation of attributes.

The special files "." and ". ." retain their accustomed semantics within the attribute hierarchy. The "." attribute file refers to the current directory and the ". ." attribute file refers to the parent directory. The unnamed directory at the head of each attribute tree is considered the "child" of the file it is associated with and the ". ." file refers to the associated file. For any non-directory file with attributes, the ". ." entry in the unnamed directory refers to a file that is not a directory.

Conceptually, the attribute model is fully general. Extended attributes can be any type of file (doors, links, directories, and so forth) and can even have their own attributes (fully recursive). As a result, the attributes associated with a file could be an arbitrarily deep directory hierarchy where each attribute could have an equally complex attribute tree associated with it. Not all implementations are able to, or want to, support the full model. Implementation are therefore permitted to reject operations that are not supported. For example, the implementation for the UFS file system allows only regular files as attributes (for example, no sub-directories) and rejects attempts to place attributes on attributes.

The following list details the operations that are rejected in the current implementation:

`link`	Any attempt to create links between attribute and non-attribute space is rejected to prevent security-related or otherwise sensitive attributes from being exposed, and therefore manipulable, as regular files.
`rename`	Any attempt to rename between attribute and non-attribute space is rejected to prevent an already linked file from being renamed and thereby circumventing the `link` restriction above.
`mkdir`, `symlink`, `mknod`	Any attempt to create a "non-regular" file in attribute space is rejected to reduce the functionality, and therefore exposure and risk, of the initial implementation.

The entire available name space has been allocated to "general use" to bring the implementation in line with the NFSv4 draft standard [NFSv4]. That standard defines "named attributes" (equivalent to Solaris Extended Attributes) with no naming restrictions. All Sun applications making use of opaque extended attributes will use the prefix "SUNW".

Shell-level API

The command interface for extended attributes is the set of applications provided by Solaris for the manipulation of attributes from the command line. This interface consists of a set of existing utilities that have been extended to be "attribute-aware", plus the `runat` utility designed to "expose" the extended attribute space so that extended attributes can be manipulated as regular files.

The `-@` option enable utilities to manipulate extended attributes. As a rule, this option enables the utility to enter into attribute space when the utility is performing a recursive traversal of file system space. This is a fully recursive concept. If the underlying file system supports recursive attributes and directory structures, the `-@` option opens these spaces to the file tree-walking algorithms.

The following utilities accommodate extended attributes (see the individual manual pages for details):

`cp` By default, `cp` ignores attributes and copies only file data. This is intended to maintain the semantics implied by `cp` currently, where attributes (such as owner and mode) are not copied unless the `-p` option is specified. With the `-@` (or `-p`) option, `cp` attempts to copy all attributes along with the file data.

`cpio` The `-@` option informs `cpio` to archive attributes, but by default `cpio` ignores extended attributes. See `Extended Archive Formats` below for a description of the new archive records.

`du` File sizes computed include the space allocated for any extended attributes present.

`find` By default, `find` ignores attributes. The `-xattr` expression provides support for searches involving attribute space. It returns true if extended attributes are present on the current file.

`fsck` The `fsck` utility manages extended attribute data on the disk. A file system with extended attributes can be mounted on versions of Solaris that are not attribute-aware (versions prior to Solaris 9), but the attributes will not be accessible and `fsck` will strip them from the files and place them in `lost+found`. Once the attributes have been stripped the file system is completely stable on Solaris versions that are not attribute-aware, but would now be considered corrupted on attribute-aware versions of Solaris. The attribute-aware `fsck` utility should be run to stabilize the file system before using it in an attribute-aware environment.

`fsdb` This `fsdb` utility is able to find the inode for the "hidden" extended attribute directory.

`ls` The `ls -@` command displays an "@" following the mode information when extended attributes are present. More precisely, the output line for a given file contains an "@" character following the mode characters if the `pathconf(2)` variable `XATTR_EXISTS` is set to true. See the `pathconf()` section below. The `-@` option uses the same general output format as the `-l` option.

`mv` When a file is moved, all attributes are carried along with the file rename. When a file is moved across a file system boundary, the copy command invoked is similar to the `cp -p` variant described above and extended attributes are "moved". If the extended file attributes cannot be replicated, the move operation fails and the source file is not removed.

`pax` The `-@` option informs `pax` to archive attributes, but by default `pax` ignores extended attributes. The `pax(1)` utility is a generic replacement for both `tar(1)` and `cpio(1)` and is able to produce either output format in its archive. See `Extended Archive Formats` below for a description of the new archive records.

`tar` In the default case, `tar` does not attempt to place attributes in the archive. If the `-@` option is specified, however, `tar` traverses into the attribute space of all files being placed in the archive and attempts to add the attributes to the archive. A new record type has been introduced for extended attribute entries in `tar` archive files (the same is true for `pax` and `cpio` archives) similar to the way ACLs records were defined. See `Extended Archive Formats` below for a description of the new archive records.

There is a class of utilities (`chmod`, `chown`, `chgrp`) that one might expect to be modified in a manner similar to those listed above. For example, one might expect that performing `chmod` on a file would not only affect the file itself but would also affect at least the extended attribute directory if not any existing extended attribute files. This is not the case. The model chosen for extended attributes implies that the attribute directory and the attributes themselves are all file objects in their own right, and can therefore have independent file status attributes associated with them (a given implementation cannot support this, for example, for intrinsic attributes). The relationship is left undefined and a fine-grained control mechanism (`runat(1)`) is provided to allow manipulation of extended attribute status attributes as necessary.

The `runat` utility has the following syntax:

`runat` *filename* [*command*]

The `runat` utility executes the supplied command in the context of the "attribute space" associated with the indicated file. If no command argument is supplied, a shell is invoked. See `runat(1)` for details.

Application-level API

The primary interface required to access extended attributes at the programmatic level is the `openat(2)` function. Once a file descriptor has been obtained for an attribute file by an `openat()` call, all normal file system semantics apply. There is no attempt to place special semantics on `read(2)`, `write(2)`, `ftruncate(3C)`, or other functions when applied to attribute file descriptors relative to "normal" file descriptors.

The set of existing attributes can be browsed by calling `openat()` with "." as the file name and the `O_XATTR` flag set, resulting in a file descriptor for the attribute directory. The list of attributes is obtained by calls to `getdents(2)` on the returned file descriptor. If the target file did not previously have any attributes associated with it, an empty top-level attribute directory

is created for the file and subsequent `getdents()` calls will return only "." and ". .". While the owner of the parent file owns the extended attribute directory, it is not charged against its quota if the directory is empty. Attribute files themselves, however, are charged against the user quota as any other regular file.

Additional system calls have been provided as convenience functions. These include the `fchownat(2)`, `fstatat(2)`, `futimesat(2)`, `renameat(2)`, `unlinkat(2)`. These new functions, along with `openat()`, provide a mechanism to access files relative to an arbitrary point in the file system, rather than only the current working directory. This mechanism is particularly useful in situations when a file descriptor is available with no path. The `openat()` function, in particular, can be used in many contexts where `chdir()` or `fchdir()` is currently required. See `chdir(2)`.

Open a file relative to a file descriptor

`int openat (int` *fd*`, const char *`*path*`, int` *oflag* `[, mode_t` *mode*`])`

The `openat(2)` function behaves exactly as `open(2)` except when given a relative path. Where `open()` resolves a relative path from the current working directory, `openat()` resolves the path based on the vnode indicated by the supplied file descriptor. When *oflag* is `O_XATTR`, `openat()` interprets the *path* argument as an extended attribute reference. The following code fragment uses `openat()` to examine the attributes of some already opened file:

```
dfd = openat(fd, ".", O_RDONLY|O_XATTR);
(void)getdents(dfd, buf, nbytes);
```

If `openat()` is passed the special value `AT_FDCWD` as its first (*fd*) argument, its behavior is identical to `open()` and the relative path arguments are interpreted relative to the current working directory. If the `O_XATTR` flag is provided to `openat()` or to `open()`, the supplied path is interpreted as a reference to an extended attribute on the current working directory.

Unlink a file relative to a directory file descriptor

`int unlinkat (int` *dirfd*`, const char *path`*flag*`, int flag`*flag*`)`

The `unlinkat(2)` function deletes an entry from a directory. The *path* argument indicates the name of the entry to remove. If *path* an absolute path, the *dirfd* argument is ignored. If it is a relative path, it is interpreted relative to the directory indicated by the *dirfd* argument. If *dirfd* does not refer to a valid directory, the function returns `ENOTDIR`. If the special value `AT_FDCWD` is specified for *dirfd*, a relative path argument is resolved relative to the current working directory. If the *flag* argument is 0, all other semantics of this function are equivalent to `unlink(2)`. If *flag* is set to `AT_REMOVEDIR`, all other semantics of this function are equivalent to `rmdir(2)`.

Rename a file relative to directories

`int renameat (int` *fromfd*`, const char *`*old*`, int` *tofd*`, const char *`*new*`)`

The renameat(2) function renames an entry in a directory, possibly moving the entry into a different directory. The *old* argument indicates the name of the entry to rename. If this argument is a relative path, it is interpreted relative to the directory indicated by the *fd* argument. If it is an absolute path, the *fromfd* argument is ignored. The *new* argument indicates the new name for the entry. If this argument is a relative path, it is interpreted relative to the directory indicated by the *tofd* argument. If it is an absolute path, the *tofd* argument is ignored.

In the relative path cases, if the directory file descriptor arguments do not refer to a valid directory, the function returns ENOTDIR. All other semantics of this function are equivalent to rename(2).

If a special value AT_FDCWD is specified for either the *fromfd* or *tofd* arguments, their associated path arguments (*old* and *new*) are interpreted relative to the current working directory if they are not specified as absolute paths. Any attempt to use renameat() to move a file that is not an extended attribute into an extended attribute directory (so that it becomes an extended attribute) will fail. The same is true for an attempt to move a file that is an extended attribute into a directory that is not an extended attribute directory.

Obtain information about a file

```
int fstatat (int fd, const char *path, struct stat* buf, int flag)
```

The fstatat(2) function obtains information about a file. If the *path* argument is relative, it is resolved relative to the *fd* argument file descriptor, otherwise the *fd* argument is ignored. If the *fd* argument is a special value AT_FDCWD the path is resolved relative to the current working directory. If the *path* argument is a null pointer, the function returns information about the file referenced by the *fd* argument. In all other relative path cases, if the *fd* argument does not refer to a valid directory, the function returns ENOTDIR. If AT_SYMLINK_NOFOLLOW is set in the *flag* argument, the function will not automatically traverse a symbolic link at the position of the path. If _AT_TRIGGER is set in the *flag* argument and the vnode is a trigger mount point, the mount is performed and the function returns the attributes of the root of the mounted filesystem. The fstatat() function is a multipurpose function that can be used in place of stat(), lstat(), or fstat(). See stat(2)

The function call stat(*path*, *buf*) is identical to fstatat(AT_FDCWD, *path*, *buf*, 0).

The function call lstat(*path*, *buf*) is identical to fstatat(AT_FDCWD, *path*, *buf*, AT_SYMLINK_NOFOLLOW)

The function call fstat(*fildes*, *buf*) is identical to fstatat(*fildes*, NULL, *buf*, 0).

Set owner and group ID

```
int fchownat (int fd, const char *path, uid_t owner, gid_t group, \
          int flag)
```

The fchownat(2) function sets the owner ID and group ID for a file. If the *path* argument is relative, it is resolved relative to the *fd* argument file descriptor, otherwise the *fd* argument is ignored. If the *fd* argument is a special value AT_FDCWD the path is resolved relative to the current working directory. If the path argument is a null pointer, the function sets the owner and group ID of the file referenced by the *fd* argument. In all other relative path cases, if the *fd* argument does not refer to a valid directory, the function returns ENOTDIR. If the *flag* argument is set to AT_SYMLINK_NOFOLLOW, the function will not automatically traverse a symbolic link at the position of the path. The fchownat() function is a multi-purpose function that can be used in place of chown(), lchown(), or fchown(). See chown(2).

The function call chown(*path*, *owner*, *group*) is equivalent to fchownat(AT_FDCWD, *path*, *owner*, *group*, 0).

The function call lchown(*path*, *owner*, *group*) is equivalent to fchownat(AT_FDCWD, *path*, *owner*, *group*, AT_SYMLINK_NOFOLLOW).

Set file access and modification times

```
int futimesat (int fd, const char *path, const struct timeval \
            times[2])
```

The futimesat(2) function sets the access and modification times for a file. If the *path* argument is relative, it is resolved relative to the *fd* argument file descriptor; otherwise the *fd* argument is ignored. If the *fd* argument is the special value AT_FDCWD, the path is resolved relative to the current working directory. If the *path* argument is a null pointer, the function sets the access and modification times of the file referenced by the *fd* argument. In all other relative path cases, if the *fd* argument does not refer to a valid directory, the function returns ENOTDIR. The futimesat() function can be used in place of utimes(2).

The function call utimes(*path*, *times*) is equivalent to futimesat(AT_FDCWD, *path*, *times*).

New pathconf() functionality

```
long int pathconf(const char *path, int name)
```

Two variables have been added to pathconf(2) to provide enhanced support for extended attribute manipulation. The XATTR_ENABLED variable allows an application to determine if attribute support is currently enabled for the file in question. The XATTR_EXISTS variable allows an application to determine whether there are any extended attributes associated with the supplied path.

Open/Create an attribute file

```
int attropen (const char *path, const char *attrpath, int oflag \
        [, mode_t mode])
```

The attropen(3C) function returns a file descriptor for the named attribute, *attrpath*, of the file indicated by *path*. The *oflag* and *mode* arguments are identical to the open(2) arguments

and are applied to the open operation on the attribute file (for example, using the `O_CREAT` flag creates a new attribute). Once opened, all normal file system operations can be used on the attribute file descriptor. The `attropen()` function is a convenience function and is equivalent to the following sequence of operations:

```
fd = open (path, O_RDONLY);
attrfd = openat(fd, attrpath, oflag|O_XATTR, mode);
close(fd);
```

The set of existing attributes can be browsed by calling `attropen()` with "." as the attribute name. The list of attributes is obtained by calling `getdents(2)` (or `fdopendir(3C)` followed by `readdir(3C)`, see below) on the returned file descriptor.

Convert an open file descriptor for a directory into a directory descriptor

```
DIR * fdopendir (const int fd)
```

The `fdopendir(3C)` function promotes a file descriptor for a directory to a directory pointer suitable for use with the `readdir(3C)` function. The originating file descriptor should not be used again following the call to `fdopendir()`. The directory pointer should be closed with a call to `closedir(3C)`. If the provided file descriptor does not reference a directory, the function returns `ENOTDIR`. This function is useful in circumstances where the only available handle on a directory is a file descriptor. See `attropen(3C)` and `openat(2)`.

Using the API

The following examples demonstrate how the API might be used to perform basic operations on extended attributes:

EXAMPLE 1 List extended attributes on a file.

```
attrdirfd = attropen("test", ".", O_RDONLY);
dirp = fdopendir(attrdirfd);
while (dp = readdir(dirp)) {
...
```

EXAMPLE 2 Open an extended attribute.

```
attrfd = attropen("test", dp->d_name, O_RDONLY);
```

or

```
attrfd = openat(attrdirfd, dp->d_name, O_RDONLY);
```

EXAMPLE 3 Read from an extended attribute.

```
while (read(attrfd, buf, 512) > 0) {
...
```

EXAMPLE 4 Create an extended attribute.

```
newfd = attropen("test", "attr", O_CREAT|O_RDWR);
```

or

```
newfd = openat(attrdirfd, "attr", O_CREAT|O_RDWR);
```

EXAMPLE 5 Write to an extended attribute.

```
count = write(newfd, buf, length);
```

EXAMPLE 6 Delete an extended attribute.

```
error = unlinkat(attrdirfd, "attr");
```

Applications intending to access the interfaces defined here as well as the POSIX and X/Open specification-conforming interfaces should define the macro `_ATFILE_SOURCE` to be 1 and set whichever feature test macros are appropriate to obtain the desired environment. See `standards(5)`.

Extended Archive Formats

As noted above in the description of command utilities modified to provide support for extended attributes, the archive formats for `tar(1)` and `cpio(1)` have been extended to provide support for archiving extended attributes. This section describes the specifics of the archive format extensions.

Extended tar format

The `tar` archive is made up of a series of 512 byte blocks. Each archived file is represented by a header block and zero or more data blocks containing the file contents. The header block is structured as shown in the following table.

Field Name	Length (in Octets)	Description
Name	100	File name string
Mode	8	12 file mode bits
Uid	8	User ID of file owner
Gid	8	Group ID of file owner
Size	12	Size of file
Mtime	12	File modification time
Chksum	8	File contents checksum
Typeflag	1	File type flag

Field Name	Length (in Octets)	Description
Linkname	100	Link target name if file linked
Magic	6	“ustar”
Version	2	“00”
Uname	32	User name of file owner
Gname	32	Group name of file owner
Devmajor	8	Major device ID if special file
Devminor	8	Minor device ID if special file
Prefix	155	Path prefix string for file

The extended attribute project extends the above header format by defining a new header type (for the `Typeflag` field). The type 'E' is defined to be used for all extended attribute files. Attribute files are stored in the `tar` archive as a sequence of two `<header ,data>` pairs. The first file contains the data necessary to locate and name the extended attribute in the file system. The second file contains the actual attribute file data. Both files use an 'E' type header. The prefix and name fields in extended attribute headers are ignored, though they should be set to meaningful values for the benefit of archivers that do not process these headers. Solaris archivers set the prefix field to “`/dev/null`” to prevent archivers that do not understand the type 'E' header from trying to restore extended attribute files in inappropriate places.

Extended cpio format

The `cpio` archive format is octet-oriented rather than block-oriented. Each file entry in the archive includes a header that describes the file, followed by the file name, followed by the contents of the file. These data are arranged as described in the following table.

`Field Name`	Length (in Octets)	Description
`c_magic`	6	70707
`c_dev`	6	First half of unique file ID
`c_ino`	6	Second half of unique file ID
`c_mode`	6	File mode bits
`c_uid`	6	User ID of file owner
`c_gid`	6	Group ID of file owner
`c_nlink`	6	Number of links referencing file
`c_rdev`	6	Information for special files

Field Name	Length (in Octets)	Description
c_mtime	11	Modification time of file
c_namesize	6	Length of file pathname
c_filesize	11	Length of file content
c_name	c_namesize	File pathname
c_filedata	c_filesize	File content

The basic archive file structure is not changed for extended attributes. The file type bits stored in the c_mode field for an attribute file are set to 0xB000. As with the tar archive format, extended attributes are stored in cpio archives as two consecutive file entries. The first file describes the location/name for the extended attribute. The second file contains the actual attribute file content. The c_name field in extended attribute headers is ignored, though it should be set to a meaningful value for the benefit of archivers that do not process these headers. Solaris archivers start the pathname with "/dev/null/"to prevent archivers that do not understand the type 'E' header from trying to restore extended attribute files in inappropriate places.

Attribute identification data format

Both the tar and cpio archive formats can contain the special files described above, always paired with the extended attribute data record, for identifying the precise location of the extended attribute. These special data files are necessary because there is no simple naming mechanism for extended attribute files. Extended attributes are not visible in the file system name space. The extended attribute name space must be "tunneled into" using the openat() function. The attribute identification data must support not only the flat naming structure for extended attributes, but also the possibility of future extensions allowing for attribute directory hierarchies and recursive attributes. The data file is therefore composed of a sequence of records. It begins with a fixed length header describing the content. The following table describes the format of this data file.

Field Name	Length (in Octets)	Description
h_version	7	Name file version
h_size	10	Length of data file
h_component_len	10	Total length of all path segments
h_link_comp_len	10	Total length of all link segments
path	h_component_len	Complex path
link_path	h_link_comp_len	Complex link path

As demonstrated above, the header is followed by a record describing the "path" to the attribute file. This path is composed of two or more path segments separated by a null character. Each segment describes a path rooted at the hidden extended attribute directory of the leaf file of the previous segment, making it possible to name attributes on attributes. The first segment is always the path to the parent file that roots the entire sequence in the normal name space. The following table describes the format of each segment.

Field Name	Length (in Octets)	Description
`h_namesz`	7	Length of segment path
`h_typeflag`	1	Actual file type of attribute file
`h_names`	`h_namesz`	Parent path + segment path

If the attribute file is linked to another file, the path record is followed by a second record describing the location of the referencing file. The structure of this record is identical to the record described above.

See Also `cp(1)`, `cpio(1)`, `find(1)`, `ls(1)`, `mv(1)`, `pax(1)`, `runat(1)`, `tar(1)`, `du(1)`, `fsck(1M)`, `chown(2)`, `link(2)`, `open(2)`, `pathconf(2)`, `rename(2)`, `stat(2)`, `unlink(2)`, `utimes(2)`, `attropen(3C)`, `standards(5)`

Name grub – GRand Unified Bootloader software on Solaris

Description The current release of the Solaris operating system is shipped with the GRUB (GRand Unified Bootloader) software. GRUB is developed and supported by the Free Software Foundation.

The overview for the GRUB Manual, accessible at `www.gnu.org`, describes GRUB:

> Briefly, a boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to an operating system kernel software (such as Linux or GNU Mach). The kernel, in turn, initializes the rest of the operating system (for example, a GNU [Ed. note: or Solaris] system).
>
> GNU GRUB is a very powerful boot loader that can load a wide variety of free, as well as proprietary, operating systems, by means of chain-loading. GRUB is designed to address the complexity of booting a personal computer; both the program and this manual are tightly bound to that computer platform, although porting to other platforms may be addressed in the future. [Ed. note: Sun has ported GRUB to the Solaris operating system.]
>
> One of the important features in GRUB is flexibility; GRUB understands filesystems and kernel executable formats, so you can load an arbitrary operating system the way you like, without recording the physical position of your kernel on the disk. Thus you can load the kernel just by specifying its file name and the drive and partition where the kernel resides.

Among Solaris machines, GRUB is supported on x86 platforms. The GRUB software that is shipped with Solaris adds two utilities not present in the open-source distribution:

`bootadm(1M)`	Enables you to manage the boot archive and make changes to the GRUB menu.
`installgrub(1M)`	Loads the boot program from disk.

Both of these utilities are described in Solaris man pages.

Beyond these two Solaris-specific utilities, the GRUB software is described in the GRUB manual, a PDF version of which is available from the Sun web site. Available in the same location is the `grub(8)` open-source man page. This man page describes the GRUB shell.

See Also `boot(1M)`, `bootadm(1M)`, `installgrub(1M)`

Solaris 10 Installation Guide: Basic Installations

System Administration Guide: Basic Administration

http://www.gnu.org/software/grub

Name gss_auth_rules – overview of GSS authorization

Description The establishment of the veracity of a user's credentials requires both authentication (Is this an authentic user?) and authorization (Is this authentic user, in fact, authorized?).

When a user makes use of Generic Security Services (GSS) versions of the `ftp` or `ssh` clients to connect to a server, the user is not necessarily authorized, even if his claimed GSS identity is authenticated, Authentication merely establishes that the user is who he says he is to the GSS mechanism's authentication system. Authorization is then required: it determines whether the GSS identity is permitted to access the specified Solaris user account.

The GSS authorization rules are as follows:

- If the mechanism of the connection has a set of authorization rules, then use those rules. For example, if the mechanism is Kerberos, then use the `krb5_auth_rules(5)`, so that authorization is consistent between raw Kerberos applications and GSS/Kerberos applications.
- If the mechanism of the connection does not have a set of authorization rules, then authorization is successful if the remote user's `gssname` matches the local user's `gssname` exactly, as compared by `gss_compare_name(3GSS)`.

Files `/etc/passwd` System account file. This information may also be in a directory service. See `passwd(4)`.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also `ftp(1)`, `ssh(1)`, `gsscred(1M)`, `gss_compare_name(3GSS)`, `passwd(4)`, `attributes(5)`, `krb5_auth_rules(5)`

Name hal – overview of hardware abstraction layer

Description The Hardware Abstraction Layer (HAL) provides a view of the various hardware attached to a system. This view is updated dynamically as hardware configuration changes by means of hotplug or other mechanisms. HAL represents a piece of hardware as a device object. A device object is identified by a unique identifer and carries a set of key/value pairs, referred to as device properties. Some properties are derived from the actual hardware, some are merged from device information files (`.fdi` files), and some are related to the actual device configuration.

HAL provides an easy-to-use API through D-Bus. D-Bus is an IPC framework that, among other features, provides a system-wide message-bus that allows applications to talk to one another. Specifically, D-Bus provides asynchronous notification such that HAL can notify other peers on the message-bus when devices are added and removed, as well as when properties on a device are changing.

In the Solaris operating system, HAL is supported by a daemon, `hald(1M)`, and a set of utilities that enable the adding and removing of devices and the modification of their properties.

See Also `hald(1M)`, `fdi(4)`

See the HAL pages, including the HAL specification, under http://freedesktop.org.

Name iconv_1250 – code set conversion tables for MS 1250 (Windows Latin 2)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
MS 1250	win2	ISO 8859-2	iso2	ISO Latin 2
MS 1250	win2	MS 852	dos2	MS-DOS Latin 2
MS 1250	win2	Mazovia	maz	Mazovia
MS 1250	win2	DHN	dhn	Dom Handlowy Nauki

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

MS 1250 to ISO 8859-2 For the conversion of MS 1250 to ISO 8859-2, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1250	**ISO 8859-2**	**MS 1250**	**ISO 8859-2**
24-211	40	235	273
212	251	236	276
213	40	237	274
214	246	241	267
215	253	245	241
216	256	246-267	40
217	254	271	261
221-231	40	273	40
232	271	274	245
233	40	276	265
234	266	247	365

MS 1250 to MS 852 For the conversion of MS 1250 to MS 852, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1250	MS 852	MS 1250	MS 852
200-211	40	311	220
212	346	312	250
213	40	313	323
214	227	314	267
215	233	315	326
216	246	316	327
217	215	317	322
220-231	40	320	321
232	347	321	343
233	40	322	325
234	230	323	340
235	234	324	342
236	247	325	212
237	253	326	231
240	377	327	236
241	363	330	374
242	364	331	336
243	235	332	351
244	317	333	353
245	244	334	232
246	40	335	355
247	365	336	335
250	371	337	341
251	40	340	352
252	270	341	240
253	256	342	203
254	252	343	307

Conversions Performed			
MS 1250	MS 852	MS 1250	MS 852
255	360	344	204
256	40	345	222
257	275	346	206
260	370	347	207
261	40	350	237
262	362	351	202
263	210	352	251
264	357	353	211
265-267	40	354	330
270	367	355	241
271	245	356	214
272	255	357	324
273	257	360	320
274	225	361	344
275	361	362	345
276	226	363	242
277	276	364	223
300	350	365	213
301	265	366	224
302	266	367	366
303	306	370	375
304	216	371	205
305	221	372	243
306	217	374	201
307	200	375	354
310	254	376	356

MS 1250 to Mazovia

For the conversion of MS 1250 to Mazovia, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1250	**Mazovia**	**MS 1250**	**Mazovia**
200-213	40	310-311	40
214	230	312	220
215-216	40	313-320	40
217	240	321	245
220-233	40	322	40
234	236	323	243
235-236	40	324-325	40
237	246	326	231
240	377	327-333	40
241-242	40	334	232
243	234	335-336	40
244	40	337	341
245	217	340-341	40
246-252	40	342	203
253	256	343	40
254	252	344	204
255-256	40	345	40
257	241	346	215
260	370	347	207
261	361	350	40
262	40	351	202
263	222	352	221
264	40	353	211
265	346	354-355	40
266	40	356	214

Conversions Performed			
MS 1250	Mazovia	MS 1250	Mazovia
267	372	357-360	40
270	40	361	244
271	206	362	40
272	40	363	242
273	257	364	223
274-276	40	365	40
277	247	366	224
300-303	40	367	366
304	216	370-373	40
305	40	374	201
306	225	375-376	40
307	200		

MS 1250 to DHN

For the conversion of MS 1250 to DHN, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1250	DHN	MS 1250	DHN
200-213	40	306	201
214	206	307-311	40
215-216	40	312	202
217	207	313-320	40
220-233	40	321	204
234	217	322	40
235-236	40	323	205
237	220	324-325	40
240	377	326	231
241-242	40	327-333	40
243	203	334	232

Conversions Performed			
MS 1250	DHN	MS 1250	DHN
244	40	335-336	40
245	200	337	341
246-252	40	340	40
253	256	341	240
254	252	342-345	40
255-256	40	346	212
257	210	347-351	40
260	370	352	213
261	361	353-354	40
262	40	355	241
263	214	356-360	40
264	40	361	215
265	346	362	40
266	40	363	216
267	372	364	223
270	40	365	40
271	211	366	224
272	40	367	366
273	257	370-371	40
274-276	40	372	243
277	221	373-376	40
300-305	40		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_1251 – code set conversion tables for MS 1251 (Windows Cyrillic)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
MS 1251	win5	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
MS 1251	win5	KOI8-R	koi8	KOI8-R
MS 1251	win5	PC Cyrillic	alt	Alternative PC Cyrillic
MS 1251	win5	Mac Cyrillic	mac	Macintosh Cyrillic

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

MS 1251 to ISO 8859-5 For the conversion of MS 1251 to ISO 8859-5, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1251	**ISO 8859-5**	**MS 1251**	**ISO 8859-5**
24	4	310	270
200	242	311	271
201	243	312	272
202	40	313	273
203	363	314	274
204-207	40	315	275
210	255	316	276
211	40	317	277
212	251	320	300
213	40	321	301
214	252	322	302
215	254	323	303
216	253	324	304
217	257	325	305

Conversions Performed			
MS 1251	ISO 8859-5	MS 1251	ISO 8859-5
220	362	326	306
221-227	40	327	307
230	255	330	310
231	40	331	311
232	371	332	312
233	40	333	313
234	372	334	314
235	374	335	315
236	373	336	316
237	377	337	317
241	256	340	320
242	376	341	321
243	250	342	322
244-247	40	343	323
250	241	344	324
251	40	345	325
252	244	346	326
253-254	40	347	327
255	55	350	330
256	40	351	331
257	247	352	332
260-261	40	353	333
262	246	354	334
263	366	355	335
264-267	40	356	336
270	361	357	337
271	360	360	340

Conversions Performed			
MS 1251	ISO 8859-5	MS 1251	ISO 8859-5
272	364	361	341
273	40	362	342
274	370	363	343
275	245	364	344
276	365	365	345
277	367	366	346
300	260	367	347
301	261	370	350
302	262	371	351
303	263	372	352
304	264	373	353
305	265	374	354
306	266	375	355
307	267	376	356

MS 1251 to KOI8-R For the conversion of MS 1251 to KOI8-R , all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1251	KOI8-R	MS 1251	KOI8-R
24	4	310	351
200	261	311	352
201	262	312	353
202	40	313	354
203	242	314	355
204-207	40	315	356
210	255	316	357
211	40	317	360
212	271	320	362

Conversions Performed			
MS 1251	KOI8-R	MS 1251	KOI8-R
213	40	321	363
214	272	322	364
215	274	323	365
216	273	324	346
217	277	325	350
220	241	326	343
221-227	40	327	376
230	255	330	373
231	40	331	375
232	251	332	377
233	40	333	371
234	252	334	370
235	254	335	374
236	253	336	340
237	257	337	361
241	276	340	301
242	256	341	302
243	270	342	327
244-247	40	343	307
250	263	344	304
251	40	345	305
252	264	346	326
253-254	40	347	332
255	55	350	311
256	40	351	312
257	267	352	313
260-261	40	353	314

Conversions Performed			
MS 1251	KOI8-R	MS 1251	KOI8-R
262	266	354	315
263	246	355	316
264-267	40	356	317
270	243	357	320
271	260	360	322
272	244	361	323
273	40	362	324
274	250	363	325
275	265	364	306
276	245	365	310
277	247	366	303
300	341	367	336
301	342	370	333
302	367	371	335
303	347	372	337
304	344	373	331
305	345	374	330
306	366	375	334
307	372	376	300

MS 1251 to PC Cyrillic

For the conversion of MS 1251 to PC Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1251	PC Cyrillic	MS 1251	PC Cyrillic
24	4	332	232
200-207	40	333	233
210	260	334	234
211-227	40	335	235

Conversions Performed			
MS 1251	PC Cyrillic	MS 1251	PC Cyrillic
230	260	336	236
231-247	40	337	237
250	360	340	240
251-254	40	341	241
255	55	342	242
256-267	40	343	243
270	361	344	244
271-277	40	345	245
300	200	346	246
301	201	347	247
302	202	350	250
303	203	351	251
304	204	352	252
305	205	353	253
306	206	354	254
307	207	355	255
310	210	356	256
311	211	357	257
312	212	360	340
313	213	361	341
314	214	362	342
315	215	363	343
316	216	364	344
317	217	365	345
320	220	366	346
321	221	367	347
322	222	370	350

Conversions Performed			
MS 1251	PC Cyrillic	MS 1251	PC Cyrillic
323	223	371	351
324	224	372	352
325	225	373	353
326	226	374	354
327	227	375	355
330	230	376	356
331	231		

MS 1251 to Mac Cyrillic

For the conversion of MS 1251 to Mac Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 1251	Mac Cyrillic	MS 1251	Mac Cyrillic
24	4	260	241
200	253	262	247
201	256	263	264
202	40	264	266
203	257	266	246
204	327	267	245
205	311	270	336
206	240	271	334
207-211	40	272	271
212	274	273	310
213	40	274	300
214	276	275	301
215	315	276	317
216	40	277	273
217	332	300	200
220	254	301	201

Conversions Performed			
MS 1251	Mac Cyrillic	MS 1251	Mac Cyrillic
221	324	302	202
222	325	303	203
223	322	304	204
224	323	305	205
225	40	306	206
226	320	307	207
227	321	310	210
230	40	311	211
231	252	312	212
232	275	313	213
233	40	314	214
234	277	315	215
235	316	316	216
236	40	317	217
237	333	320	220
240	312	321	221
241	330	322	222
242	331	323	223
243	267	324	224
244	377	325	225
245	242	326	226
246	40	327	227
247	244	330	230
250	335	331	231
252	270	332	232
253	307	333	233
254	302	334	234

Conversions Performed			
MS 1251	Mac Cyrillic	MS 1251	Mac Cyrillic
255	55	335	235
256	250	336	236
257	272	337	237
355	316		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv – code set conversion tables

Description The following code set conversions are supported:

Code Set Conversions Supported

Code	Symbol	Target Code	Symbol	Target Output
ISO 646	646	ISO 8859-1	8859	US ASCII
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English ASCII
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English ASCII
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish
ISO 8859-16	iso16	ISO 8859-2	iso2	ISO Latin 2
ISO 8859-2	iso2	ISO 8859-16	iso16	ISO Latin 10
ISO 8859-16	iso16	IBM 850	ibm850	IBM 850 code page
ISO 8859-16	iso16	IBM 870	ibm870	IBM 870 code page
ISO 8859-2	iso2	MS 1250	win2	Windows Latin 2
ISO 8859-2	iso2	MS 852	dos2	MS-DOS Latin 2
ISO 8859-2	iso2	Mazovia	maz	Mazovia
IBM 850	ibm850	ISO 8859-16	iso16	ISO Latin 10
IBM 870	ibm870	ISO 8859-16	iso16	ISO Latin 10
MS 1250	win2	DHN	dhn	Dom Handlowy Nauki
MS 852	dos2	ISO 8859-2	iso2	ISO Latin 2
MS 852	dos2	MS 1250	win2	Windows Latin 2
MS 852	dos2	Mazovia	maz	Mazovia
MS 852	dos2	DHN	dhn	Dom Handlowy Nauki
Mazovia	maz	ISO 8859-2	iso2	ISO Latin 2
Mazovia	maz	MS 1250	win2	Windows Latin 2
Mazovia	maz	MS 852	dos2	MS-DOS Latin 2
Mazovia	maz	DHN	dhn	Dom Handlowy Nauki
DHN	dhn	ISO 8859-2	iso2	ISO Latin 2
DHN	dhn	MS 1250	win2	Windows Latin 2
DHN	dhn	MS 852	dos2	MS-DOS Latin 2
DHN	dhn	Mazovia	maz	Mazovia
ISO 8859-5	iso5	KOI8-R	koi8	KOI8-R
ISO 8859-5	iso5	PC Cyrillic	alt	Alternative PC Cyrillic

```
ISO 8859-5      iso5     MS 1251        win5     Windows Cyrillic
ISO 8859-5      iso5     Mac Cyrillic   mac      Macintosh Cyrillic
KOI8-R          koi8     ISO 8859-5     iso5     ISO 8859-5 Cyrillic
KOI8-R          koi8     PC Cyrillic    alt      Alternative PC Cyrillic
KOI8-R          koi8     MS 1251        win5     Windows Cyrillic
KOI8-R          koi8     Mac Cyrillic   mac      Macintosh Cyrillic
PC Cyrillic     alt      ISO 8859-5     iso5     ISO 8859-5 Cyrillic
PC Cyrillic     alt      KOI8-R         koi8     KOI8-R
PC Cyrillic     alt      MS 1251        win5     Windows Cyrillic
PC Cyrillic     alt      Mac Cyrillic   mac      Macintosh Cyrillic
MS 1251         win5     ISO 8859-5     iso5     ISO 8859-5 Cyrillic
MS 1251         win5     KOI8-R         koi8     KOI8-R
MS 1251         win5     PC Cyrillic    alt      Alternative PC Cyrillic
MS 1251         win5     Mac Cyrillic   mac      Macintosh Cyrillic
Mac Cyrillic    mac      ISO 8859-5     iso5     ISO 8859-5 Cyrillic
Mac Cyrillic    mac      KOI8-R         koi8     KOI8-R
Mac Cyrillic    mac      PC Cyrillic    alt      Alternative PC Cyrillic
Mac Cyrillic    mac      MS 1251        win5     Windows Cyrillic
```

Conversions The conversions are performed according to the tables contained in the manual pages cross-referenced in the `Index of Conversion Code Tables` below.

Index of Conversion Code Tables		
Code	Target Code	See Manual Page
ISO 646	ISO 8859-1	iconv_646 (5)
ISO 646de	ISO 8859-1	
ISO 646da	ISO 8859-1	
ISO 646en	ISO 8859-1	
ISO 646es	ISO 8859-1	
ISO 646fr	ISO 8859-1	
ISO 646it	ISO 8859-1	
ISO 646sv	ISO 8859-1	
ISO 8859-1	ISO 646	iconv_8859-1 (5)
ISO 8859-1	ISO 646de	
ISO 8859-1	ISO 646da	
ISO 8859-1	ISO 646en	
ISO 8859-1	ISO 646es	

Index of Conversion Code Tables		
ISO 8859-1	ISO 646fr	
ISO 8859-1	ISO 646it	
ISO 8859-1	ISO 646sv	
ISO 8859-2	MS 1250	iconv_8859-2 (5)
ISO 8859-2	MS 852	
ISO 8859-2	Mazovia	
ISO 8859-2	DHN	
MS 1250	ISO 8859-2	iconv_1250 (5)
MS 1250	MS 852	
MS 1250	Mazovia	
MS 1250	DHN	
MS 852	ISO 8859-2	iconv_852 (5)
MS 852	MS 1250	
MS 852	Mazovia	
MS 852	DHN	
Mazovia	ISO 8859-2	iconv_maz (5)
Mazovia	MS 1250	
Mazovia	MS 852	
Mazovia	DHN	

Index of Conversion Code Tables		
Code	Target Code	See Manual Page
DHN	ISO 8859-2	iconv_dhn (5)
DHN	MS 1250	
DHN	MS 852	
DHN	Mazovia	
ISO 8859-5	KOI8-R	iconv_8859-5 (5)
ISO 8859-5	PC Cyrillic	

Index of Conversion Code Tables		
ISO 8859-5	MS 1251	
ISO 8859-5	Mac Cyrillic	
KOI8-R	ISO 8859-5	iconv_koi8-r (5)
KOI8-R	PC Cyrillic	
KOI8-R	MS 1251	
KOI8-R	Mac Cyrillic	
PC Cyrillic	ISO 8859-5	iconv_pc_cyr (5)
PC Cyrillic	KOI8-R	
PC Cyrillic	MS 1251	
PC Cyrillic	Mac Cyrillic	
MS 1251	ISO 8859-5	iconv_1251 (5)
MS 1251	KOI8-R	
MS 1251	PC Cyrillic	
MS 1251	Mac Cyrillic	
Mac Cyrillic	ISO 8859-5	iconv_mac_cyr (5)
Mac Cyrillic	KOI8-R	
Mac Cyrillic	PC Cyrillic	
Mac Cyrillic	MS 1251	

Files

`/usr/lib/iconv/*.so`
conversion modules

`/usr/lib/iconv/*.t`
Conversion tables.

`/usr/lib/iconv/geniconvtbl/binarytables/*.bt`
Conversion binary tables.

`/usr/lib/iconv/iconv_data`
List of conversions supported by conversion tables.

See Also `iconv(1)`, `iconv(3C)`, `iconv_1250(5)`, `iconv_1251(5)`, `iconv_646(5)`, `iconv_852(5)`, `iconv_8859-1(5)`, `iconv_8859-2(5)`, `iconv_8859-5(5)`, `iconv_dhn(5)`, `iconv_koi8-r(5)`, `iconv_mac_cyr(5)`, `iconv_maz(5)`, `iconv_pc_cyr(5)`, `iconv_unicode(5)`

Name iconv_646 – code set conversion tables for ISO 646

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
ISO 646	646	ISO 8859-1	8859	US ASCII
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English ASCII
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

ISO 646 (US ASCII) to ISO 8859-1

For the conversion of ISO 646 to ISO 8859-1, all characters in ISO 646 can be mapped unchanged to ISO 8859-1

ISO 646de (GERMAN) to ISO 8859-1

For the conversion of ISO 646de to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646de	**ISO 8859-1**	**ISO 646de**	**ISO 8859-1**
100	247	173	344
133	304	174	366
134	326	175	374
135	334	176	337

ISO 646da (DANISH) to ISO 8859-1

For the conversion of ISO 646da to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646da	ISO 8859-1	ISO 646da	ISO 8859-1
133	306	173	346
134	330	174	370
135	305	175	345

ISO 646en (ENGLISH ASCII) to ISO 8859-1

For the conversion of ISO 646en to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646en	ISO 8859-1
043	243

ISO 646es (SPANISH) to ISO 8859-1

For the conversion of ISO 646es to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646es	ISO 8859-1	ISO 646es	ISO 8859-1
100	247	173	260
133	241	174	361
134	321	175	347
135	277		

ISO 646fr (FRENCH) to ISO 8859-1

For the conversion of ISO 646fr to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646fr	ISO 8859-1	ISO 646fr	ISO 8859-1
043	243	173	351
100	340	174	371
133	260	175	350
134	347	176	250
135	247		

ISO 646it (ITALIAN) to ISO 8859-1

For the conversion of ISO 646it to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646it	ISO 8859-1	ISO 646it	ISO 8859-1
043	243	140	371
100	247	173	340
133	260	174	362
134	347	175	350
135	351	176	354

ISO 646sv (SWEDISH) to ISO 8859-1

For the conversion of ISO 646sv to ISO 8859-1, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 646sv	ISO 8859-1	ISO 646sv	ISO 8859-1
100	311	140	351
133	304	173	344
134	326	174	366
135	305	175	345
136	334	176	374

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_852 – code set conversion tables for MS 852 (MS-DOS Latin 2)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
MS 852	dos2	ISO 8859-2	iso2	ISO Latin 2
MS 852	dos2	MS 1250	win2	Windows Latin 2
MS 852	dos2	Mazovia	maz	Mazovia
MS 852	dos2	DHN	dhn	Dom Handlowy Nauki

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

MS 852 to ISO 8859-2 For the conversion of MS 852 to ISO 8859-2, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 852	**ISO 8859-2**	`MS 852`	**ISO 8859-2**
24-177	40	271-274	40
200	307	275	257
201	374	276	277
202	351	277-305	40
203	342	306	303
204	344	307	343
205	371	310-316	40
206	346	317	244
207	347	320	360
210	263	321	320
211	353	322	317
212	325	323	313
213	365	324	357
214	356	325	322

Conversions Performed			
MS 852	**ISO 8859-2**	MS 852	**ISO 8859-2**
215	254	326	315
216	304	327	316
217	306	330	354
220	311	331-334	40
221	305	335	336
222	345	336	331
223	364	337	40
224	366	340	323
225	245	341	337
226	265	342	324
227	246	343	321
230	266	344	361
231	326	345	362
232	334	346	251
233	253	347	271
234	273	350	300
235	243	351	332
236	327	352	340
237	350	353	333
240	341	354	375
241	355	355	335
242	363	356	376
243	372	357	264
244	241	360	255
245	261	361	275
246	256	362	262
247	276	363	267

Conversions Performed			
MS 852	**ISO 8859-2**	MS 852	**ISO 8859-2**
250	312	364	242
251	352	365	247
252	40	366	367
253	274	367	270
254	310	370	260
255	272	371	250
256-264	40	372	377
265	301	374	330
266	302	375	370
267	314	376	40
270	252		

MS 852 to MS 1250

For the conversion of MS 852 to MS 1250, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 852	**MS 1250**	**MS 852**	**MS 1250**
200	307	270	252
201	374	271-274	40
202	351	275	257
203	342	276	277
204	344	277-305	40
205	371	306	303
206	346	307	343
207	347	310-316	40
210	263	317	244
211	353	320	360
212	325	321	320
213	365	322	317

Conversions Performed			
MS 852	MS 1250	MS 852	MS 1250
214	356	323	313
215	217	324	357
216	304	325	322
217	306	326	315
220	311	327	316
221	305	330	354
222	345	331-334	40
223	364	335	336
224	366	336	331
225	274	337	40
226	276	340	323
227	214	341	337
230	234	342	324
231	326	343	321
232	334	344	361
233	215	345	362
234	235	346	212
235	243	347	232
236	327	350	300
237	350	351	332
240	341	352	340
241	355	353	333
242	363	354	375
243	372	355	335
244	245	356	376
245	271	357	264
246	216	360	255

Conversions Performed			
MS 852	MS 1250	MS 852	MS 1250
247	236	361	275
250	312	362	262
251	352	363	241
252	254	364	242
253	237	365	247
254	310	366	367
255	272	367	270
256	253	370	260
257	273	371	250
260-264	40	372	377
265	301	374	330
266	302	375	370
267	314	376	40

MS 852 to Mazovia

For the conversion of MS 852 to Mazovia, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 852	Mazovia	MS 852	Mazovia
205	40	246-247	40
206	215	250	220
210	222	251	221
212-213	40	253	246
215	240	254-270	40
217	225	275	241
220-226	40	276	247
227	230	306-336	40
230	236	340	243
233-234	40	342	40

Conversions Performed			
MS 852	Mazovia	MS 852	Mazovia
235	234	343	245
236-243	40	344	244
244	217	345-375	40
245	206		

MS 852 to DHN

For the conversion of MS 852 to DHN, all characters not in the following table are mapped unchanged.

Conversions Performed			
MS 852	DHN	MS 852	DHN
200-205	40	244	200
206	212	245	211
207	40	246-247	40
210	214	250	202
211-214	40	251	213
215	207	253	220
216	40	254-270	40
217	201	275	210
220-226	40	276	221
227	206	306-336	40
230	217	340	205
233-234	40	342	40
235	203	343	204
236-237	40	344	215
242	216	345-375	40
252	254		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also iconv(1), iconv(3C), iconv(5)

Name iconv_8859-1 – code set conversion tables for ISO 8859-1 (Latin 1)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
ISO 8859-1	8859	ISO 646	646	7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English ASCII
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

ISO 8859-1 to ISO 646 (7-bit ASCII) For the conversion of ISO 8859-1 to ISO 646, all characters not in the following table are mapped unchanged.

```
Converted to Underscore '_' (137)
---------------------------------
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377
```

ISO 8859-1 to ISO 646de (GERMAN)

For the conversion of ISO 8859-1 to ISO 646de, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646de	ISO 8859-1	ISO 646de
247	100	337	176
304	133	344	173
326	134	366	174
334	135	374	175

```
Converted to Underscore '_' (137)
---------------------------------
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303     305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325     327
330 331 332 333     335 336 337
340 341 342 343     345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365     367
370 371 372 373     375 376 377
```

ISO 8859-1 to ISO 646da (DANISH)

For the conversion of ISO 8859-1 to ISO 646da, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646da	ISO 8859-1	ISO 646da
305	135	345	175
306	133	346	173
330	134	370	174

```
Converted to Underscore '_' (137)
---------------------------------
133 134 135 173 174 175
```

```
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304         307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
    331 332 333 334 335 336 337
340 341 342 343 344         347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
371 372 373 374         376 377
```

ISO 8859-1 to ISO 646en (ENGLISH ASCII)

For the conversion of ISO 8859-1 to ISO 646en, all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	**ISO 646en**
243	043

```
Converted to Underscore '_' (137)
---------------------------------
043
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242     244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377
```

ISO 8859-1 to ISO 646fr (FRENCH)

For the conversion of ISO 8859-1 to ISO 646fr, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646fr	ISO 8859-1	ISO 646fr
243	043	347	134
247	135	350	175
250	176	351	173
260	133	371	174
340	100		

```
Converted to Underscore '_' (137)
---------------------------------
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242     244 245 246
    251 252 253 254 255 256 257
    261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
    341 342 343 344 345 346
        352 353 354 355 356 357
360 361 362 363 364 365 366 367
370     372 373 374 375 376 377
```

ISO 8859-1 to ISO 646it (ITALIAN)

For the conversion of ISO 8859-1 to ISO 646it, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646it	ISO 8859-1	ISO 646it
243	043	350	175
247	100	351	135
260	133	354	176

Conversions Performed			
ISO 8859-1	ISO 646it	ISO 8859-1	ISO 646it
340	173	362	174
347	134	371	140

```
Converted to Underscore '_' (137)
---------------------------------
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242     244 245 246
250 251 252 253 254 255 256 257
    261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
    341 342 343 344 345 346
        352 353 354 355 356 357
360 361 362 363 364 365 366 367
370     372 373 374 375 376 377
```

ISO 8859-1 to ISO 646es (SPANISH)

For the conversion of ISO 8859-1 to ISO 646es, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646es	ISO 8859-1	ISO 646es
241	133	321	134
247	100	347	175
260	173	361	174
277	135		

```
Converted to Underscore '_' (137)
---------------------------------
100 133 134 135 173 174 175
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
```

```
240     242 243 244 245 246
250 251 252 253 254 255 256 257
    261 262 263 264 265 266 267
270 271 272 273 274 275 276
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320     322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346
350 351 352 353 354 355 356 357
360     362 363 364 365 366 367
370 371 372 373 374 375 376 377
```

ISO 8859-1 to ISO 646sv (SWEDISH)

For the conversion of ISO 8859-1 to ISO 646sv, all characters not in the following tables are mapped unchanged.

Conversions Performed			
ISO 8859-1	ISO 646sv	ISO 8859-1	ISO 646sv
304	133	344	173
305	135	345	175
311	100	351	140
326	134	366	174
334	136	374	176

```
Converted to Underscore '_' (137)
---------------------------------
100 133 134 135 136 140
173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303         306 307
310     312 313 314 315 316 317
320 321 322 323 324 325     327
330 331 332 333     335 336 337
340 341 342 343         346 347
350     352 353 354 355 356 357
360 361 362 363 364 365     367
370 371 372 373     375 376 377
```

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_8859-2 – code set conversion tables for ISO 8859-2 (Latin 2)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
ISO 8859-2	iso2	MS 1250	win2	Windows Latin 2
ISO 8859-2	iso2	MS 852	dos2	MS-DOS Latin 2
ISO 8859-2	iso2	Mazovia	maz	Mazovia
ISO 8859-2	iso2	DHN	dhn	Dom Handlowy Nauki

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

ISO 8859-2 to MS 1250 For the conversion of ISO 8859-2 to MS 1250, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-2	**MS 1250**	**ISO 8859-2**	**MS 1250**
24	4	261	271
177-237	40	265	276
241	245	266	234
245	274	267	241
246	214	271	232
251	212	273	235
253	215	274	237
254	217	276	236
256	216	266	236

ISO 8859-2 to MS 852 For the conversion of ISO 8859-2 to MS 852, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-2	MS 852	ISO 8859-2	MS 852
24	4	316	327
177-237	40	317	322
240	377	320	321
241	244	321	343
242	364	322	325
243	235	323	340
244	317	324	342
245	225	325	212
246	227	326	231
247	365	327	236
250	371	330	374
251	346	331	336
252	270	332	351
253	233	333	353
254	215	334	232
255	360	335	355
256	246	336	335
257	275	337	341
260	370	340	352
261	245	341	240
262	362	342	203
263	210	343	307
264	357	344	204
265	226	345	222
266	230	346	206
267	363	347	207
270	367	350	237

Conversions Performed			
ISO 8859-2	MS 852	ISO 8859-2	MS 852
271	347	351	202
272	255	352	251
273	234	353	211
274	253	354	330
275	361	355	241
276	247	356	214
277	276	357	324
300	350	360	320
301	265	361	344
302	266	362	345
303	306	363	242
304	216	364	223
305	221	365	213
306	217	366	224
307	200	367	366
310	254	370	375
311	220	371	205
312	250	372	243
313	323	374	201
314	267	375	354
315	326	376	356
366	367		

ISO 8859-2 to Mazovia

For the conversion of ISO 8859-2 to Mazovia, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-2	Mazovia	ISO 8859-2	Mazovia
24	4	323	243

Conversions Performed			
ISO 8859-2	Mazovia	ISO 8859-2	Mazovia
177-237	40	324-325	40
240	377	326	231
241	217	327-333	40
242	40	334	232
243	234	335-336	40
244-245	40	337	341
246	230	340-341	40
247-253	40	342	203
254	240	343	40
255-256	40	344	204
257	241	345	40
260	370	346	215
261	206	347	207
262	40	350	40
263	222	351	202
264-265	40	352	221
266	236	353	211
267-273	40	354-355	40
274	246	356	214
275-276	40	357-360	40
277	247	361	244
300-303	40	362	40
304	216	363	242
305	40	364	223
306	225	365	40
307	200	366	224
310-311	40	367	366

Conversions Performed			
ISO 8859-2	Mazovia	ISO 8859-2	Mazovia
312	220	370-373	40
313-320	40	374	201
321	245	375-376	40
322	40		

ISO 8859-2 to DHN

For the conversion of ISO 8859-2 to DHN, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-2	DHN	ISO 8859-2	DHN
24	4	322	40
177-237	40	323	205
240	377	324-325	40
241	200	326	231
242	40	327-333	40
243	203	334	232
244-245	40	335-336	40
246	206	337	341
247-253	40	340	40
254	207	341	240
255-256	40	342-345	40
257	210	346	212
260	370	347-351	40
261	211	352	213
262	40	353-354	40
263	214	355	241
264-265	40	356-360	40
266	217	361	215
267-273	40	362	40

Conversions Performed			
ISO 8859-2	DHN	ISO 8859-2	DHN
274	220	363	216
275-276	40	364	223
277	221	365	40
300-305	40	366	224
306	201	367	366
307-311	40	370-371	40
312	202	372	243
313-320	40	373-376	40
321	204		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_8859-5 – code set conversion tables for ISO 8859-5 (Cyrillic)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
ISO 8859-5	iso5	KOI8-R	koi8	KOI8-R
ISO 8859-5	iso5	PC Cyrillic	alt	Alternative PC Cyrillic
ISO 8859-5	iso5	MS 1251	win5	Windows Cyrillic
ISO 8859-5	iso5	Mac Cyrillic	mac	Macintosh Cyrillic

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

ISO 8859-5 to KOI8-R For the conversion of ISO 8859-5 to KOI8-R, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-5	**KOI8-R**	**ISO 8859-5**	**KOI8-R**
24	4	320	301
241	263	321	302
242	261	322	327
243	262	323	307
244	264	324	304
245	265	325	305
246	266	327	332
247	267	330	311
250	270	331	312
251	271	332	313
252	272	333	314
253	273	334	315
254	274	335	316
256	276	336	317

Conversions Performed			
ISO 8859-5	KOI8-R	ISO 8859-5	KOI8-R
257	277	337	320
260	341	340	322
261	342	341	323
262	367	342	324
263	347	343	325
264	344	344	306
265	345	345	310
266	366	346	303
267	372	347	336
270	351	350	333
271	352	351	335
272	353	352	337
273	354	353	331
274	355	354	330
275	356	355	334
276	357	356	300
277	360	357	321
300	362	360	260
301	363	361	243
302	364	362	241
303	365	363	242
304	346	364	244
305	350	365	245
306	343	366	246
307	376	367	247
310	373	370	250
311	375	371	251

Conversions Performed			
ISO 8859-5	KOI8-R	ISO 8859-5	KOI8-R
312	377	372	252
313	371	373	253
314	370	374	254
315	374	375	255
316	340	376	256
317	361		

ISO 8859-5 to PC Cyrillic

For the conversion of ISO 8859-5 to PC Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-5	PC Cyrillic	ISO 8859-5	PC Cyrillic
24	4	307	227
200-240	40	310	230
241	360	311	231
242-254	40	312	232
255	260	313	233
256-257	40	314	234
260	200	315	235
261	201	316	236
262	202	317	237
263	203	320	240
264	204	321	241
265	205	322	242
266	206	323	243
267	207	324	244
270	210	325	245
271	211	326	246
272	212	327	247

Conversions Performed			
ISO 8859-5	PC Cyrillic	ISO 8859-5	PC Cyrillic
273	213	330	250
274	214	331	251
275	215	332	252
276	216	333	253
277	217	334	254
300	220	335	255
301	221	336	256
302	222	337	257
303	223	360-374	40
304	224	375	260
305	225	376	40
306	226	365	40

ISO 8859-5 to MS 1251

For the conversion of ISO 8859-5 to MS 1251, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-5	MS 1251	ISO 8859-5	MS 1251
24	4	317	337
200-237	40	320	340
241	250	321	341
242	200	322	342
243	201	323	343
244	252	324	344
245	275	325	345
246	262	326	346
247	257	327	347
250	243	330	350
251	212	331	351

Conversions Performed			
ISO 8859-5	MS 1251	ISO 8859-5	MS 1251
252	214	332	352
253	216	333	353
254	215	334	354
255	210	335	355
256	241	336	356
257	217	337	357
260	300	340	360
261	301	341	361
262	302	342	362
263	303	343	363
264	304	344	364
265	305	345	365
266	306	346	366
267	307	347	367
270	310	350	370
271	311	351	371
272	312	352	372
273	313	353	373
274	314	354	374
275	315	355	375
276	316	356	376
277	317	357	377
300	320	360	271
301	321	361	270
302	322	362	220
303	323	363	203
304	324	364	272

Conversions Performed			
ISO 8859-5	MS 1251	ISO 8859-5	MS 1251
305	325	365	276
306	326	366	263
307	327	367	277
310	330	370	274
311	331	371	232
312	332	372	234
313	333	373	236
314	334	374	235
315	335	375	210
316	336	376	242
376	331		

ISO 8859-5 to Mac Cyrillic

For the conversion of ISO 8859-5 to Mac Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
ISO 8859-5	Mac Cyrillic	ISO 8859-5	Mac Cyrillic
24	4	317	237
200-237	40	320	340
240	312	321	341
241	335	322	342
242	253	323	343
243	256	324	344
244	270	325	345
245	301	326	346
246	247	327	347
247	272	330	350
250	267	331	351
251	274	332	352

Conversions Performed			
ISO 8859-5	Mac Cyrillic	ISO 8859-5	Mac Cyrillic
252	276	333	353
253	40	334	354
254	315	335	355
255	40	336	356
256	330	337	357
257	332	340	360
260	200	341	361
261	201	342	362
262	202	343	363
263	203	344	364
264	204	345	365
265	205	346	366
266	206	347	367
267	207	350	370
270	210	351	371
271	211	352	372
272	212	353	373
273	213	354	374
274	214	355	375
275	215	356	376
276	216	357	337
277	217	360	334
300	220	361	336
301	221	362	254
302	222	363	257
303	223	364	271
304	224	365	317

Conversions Performed			
ISO 8859-5	Mac Cyrillic	ISO 8859-5	Mac Cyrillic
305	225	366	264
306	226	367	273
307	227	370	300
310	230	371	275
311	231	372	277
312	232	373	40
313	233	374	316
314	234	375	40
315	235	376	331
316	236		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_dhn – code set conversion tables for DHN (Dom Handlowy Nauki)

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
DHN	dhn	ISO 8859-2	iso2	ISO Latin 2
DHN	dhn	MS 1250	win2	Windows Latin 2
DHN	dhn	MS 852	dos2	MS-DOS Latin 2
DHN	dhn	Mazovia	maz	Mazovia

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

DHN to ISO 8859-2 For the conversion of DHN to ISO 8859-2, all characters not in the following table are mapped unchanged.

Conversions Performed			
DHN	**ISO 8859-2**	**DHN**	**ISO 8859-2**
24-177	40	222	40
200	241	223	364
201	306	224	366
202	312	225-230	40
203	243	231	326
204	321	232	334
205	323	233-237	40
206	246	240	341
207	254	241	355
210	257	242	363
211	261	243	372
212	346	244-340	40
213	352	341	337
214	263	342-365	40

Conversions Performed			
DHN	ISO 8859-2	DHN	ISO 8859-2
215	361	366	367
216	363	367	40
217	266	370	260
220	274	371-376	40
221	277		

DHN to MS 1250

For the conversion of DHN to MS 1250, all characters not in the following table are mapped unchanged.

Conversions Performed			
DHN	MS 1250	DHN	MS 1250
200	245	233-237	40
201	306	240	341
202	312	241	355
203	243	242	363
204	321	243	372
205	323	244-251	40
206	214	252	254
207	217	253-255	40
210	257	256	253
211	271	257	273
212	346	260-340	40
213	352	341	337
214	263	342-345	40
215	361	346	265
216	363	347-360	40
217	234	361	261
220	237	362-365	40
221	277	366	367

Conversions Performed			
DHN	MS 1250	DHN	MS 1250
222	40	367	40
223	364	370	260
224	366	371	40
225-230	40	372	267
231	326	373-376	40
232	334		

DHN to MS 852

For the conversion of DHN to MS 852, all characters not in the following table are mapped unchanged.

Conversions Performed			
DHN	MS 852	DHN	MS 852
200	244	212	206
201	217	213	251
202	250	214	210
203	235	215	344
204	343	216	242
205	340	217	230
206	227	220	253
207	215	221	276
210	275	222-375	40
211	245		

DHN to Mazovia

For the conversion of DHN to Mazovia, all characters not in the following table are mapped unchanged.

Conversions Performed			
DHN	Mazovia	DHN	Mazovia
200	217	212	215
201	225	213	221

Conversions Performed			
DHN	Mazovia	DHN	Mazovia
202	220	214	222
203	234	215	244
204	245	216	242
205	243	217	236
206	230	220	246
207	240	221	247
210	241	222-247	40
211	206		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_koi8-r – code set conversion tables for KOI8-R

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
KOI8-R	koi8	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
KOI8-R	koi8	PC Cyrillic	alt	Alternative PC Cyrillic
KOI8-R	koi8	MS 1251	win5	Windows Cyrillic
KOI8-R	koi8	Mac Cyrillic	mac	Macintosh Cyrillic

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

KOI8-R to ISO 8859-5 For the conversion of KOI8-R to ISO 8859-5, all characters not in the following table are mapped unchanged.

Conversions Performed			
KOI8-R	**ISO 8859-5**	**KOI8-R**	**ISO 8859-5**
24	4	320	337
241	362	321	357
242	363	322	340
243	361	323	341
244	364	324	342
245	365	325	343
246	366	327	322
247	367	330	354
250	370	331	353
251	371	332	327
252	372	333	350
253	373	334	355
254	374	335	351
256	376	336	347

Conversions Performed			
KOI8-R	ISO 8859-5	KOI8-R	ISO 8859-5
257	377	337	352
260	360	340	316
261	242	341	260
262	243	342	261
263	241	343	306
264	244	344	264
265	245	345	265
266	246	346	304
267	247	347	263
270	250	350	305
271	251	351	270
272	252	352	271
273	253	353	272
274	254	354	273
275	255	355	274
276	256	356	275
277	257	357	276
300	356	360	277
301	320	361	317
302	321	362	300
303	346	363	301
304	324	364	302
305	325	365	303
306	344	366	266
307	323	367	262
310	345	370	314
311	330	371	313

Conversions Performed			
KOI8-R	ISO 8859-5	KOI8-R	ISO 8859-5
312	331	372	267
313	332	373	310
314	333	374	315
315	334	375	311
316	335	376	307
317	336		

KOI8-R to PC Cyrillic

For the conversion of KOI8-R to PC Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
KOI8-R	PC Cyrillic	KOI8-R	PC Cyrillic
24	4	333	350
200-242	40	334	355
243	361	335	351
244-254	40	336	347
255	260	337	352
256-262	40	340	236
263	360	341	200
264-274	40	342	201
275	260	343	226
276-277	40	344	204
300	356	345	205
301	240	346	224
302	241	347	203
303	346	350	225
304	244	351	210
305	245	352	211
306	344	353	212

Conversions Performed			
KOI8-R	PC Cyrillic	KOI8-R	PC Cyrillic
307	243	354	213
310	345	355	214
311	250	356	215
312	251	357	216
313	252	360	217
314	253	361	237
315	254	362	220
316	255	363	221
317	256	364	222
320	257	365	223
321	357	366	206
322	340	367	202
323	341	370	234
324	342	371	233
325	343	372	207
326	246	373	230
327	242	374	235
330	354	375	231
331	353	376	227
332	247		

KOI8-R to MS 1251

For the conversion of KOI8-R to MS 1251, all characters not in the following table are mapped unchanged.

Conversions Performed			
KOI8-R	MS 1251	KOI8-R	MS 1251
24	4	317	356
200-237	40	320	357
241	220	321	377

Conversions Performed			
KOI8-R	MS 1251	KOI8-R	MS 1251
242	203	322	360
243	270	323	361
244	272	324	362
245	276	325	363
246	263	326	346
247	277	327	342
250	274	330	374
251	232	331	373
252	234	332	347
253	236	333	370
254	235	334	375
255	210	335	371
256	242	336	367
257	237	337	372
260	271	340	336
261	200	341	300
262	201	342	301
263	250	343	326
264	252	344	304
265	275	345	305
266	262	346	324
267	257	347	303
270	243	350	325
271	212	351	310
272	214	352	311
273	216	353	312
274	215	354	313

Conversions Performed			
KOI8-R	MS 1251	KOI8-R	MS 1251
275	210	355	314
276	241	356	315
277	217	357	316
300	376	360	317
301	340	361	337
302	341	362	320
303	366	363	321
304	344	364	322
305	345	365	323
306	364	366	306
307	343	367	302
310	365	370	334
311	350	371	333
312	351	372	307
313	352	373	330
314	353	374	335
315	354	375	331
316	355	376	327
376	227		

KOI8-R to Mac Cyrillic

For the conversion of KOI8-R to Mac Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
KOI8-R	Mac Cyrillic	KOI8-R	Mac Cyrillic
24	4	317	356
200-237	40	320	357
240	312	321	337
241	254	322	360

Conversions Performed			
KOI8-R	Mac Cyrillic	KOI8-R	Mac Cyrillic
242	257	323	361
243	336	324	362
244	271	325	363
245	317	326	346
246	264	327	342
247	273	330	374
250	300	331	373
251	275	332	347
252	277	333	370
253	40	334	375
254	316	335	371
255	40	336	367
256	331	337	372
257	333	340	236
260	334	341	200
261	253	342	201
262	256	343	226
263	335	344	204
264	270	345	205
265	301	346	224
266	247	347	203
267	272	350	225
270	267	351	210
271	274	352	211
272	276	353	212
273	40	354	213
274	315	355	214

Conversions Performed			
KOI8-R	Mac Cyrillic	KOI8-R	Mac Cyrillic
275	40	356	215
276	330	357	216
277	332	360	217
300	376	361	237
301	340	362	220
302	341	363	221
303	366	364	222
304	344	365	223
305	345	366	206
306	364	367	202
307	343	370	234
310	365	371	233
311	350	372	207
312	351	373	230
313	352	374	235
314	353	375	231
315	354	376	227
316	355		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_mac_cyr – code set conversion tables for Macintosh Cyrillic

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
Mac Cyrillic	mac	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
Mac Cyrillic	mac	KOI8-R	koi8	KOI8-R
Mac Cyrillic	mac	PC Cyrillic	alt	Alternative PC Cyrillic
Mac Cyrillic	mac	MS 1251	win5	Windows Cyrillic

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

Mac Cyrillic to ISO 8859-5 For the conversion of Mac Cyrillic to ISO 8859-5, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mac Cyrillic	**ISO 8859-5**	**Mac Cyrillic**	**ISO 8859-5**
24	4	276	252
200	260	277	372
201	261	300	370
202	262	301	245
203	263	302-311	40
204	264	312	240
205	265	313	242
206	266	314	362
207	267	315	254
210	270	316	374
211	271	317	365
212	272	320-327	40
213	273	330	256
214	274	331	376

Conversions Performed			
Mac Cyrillic	ISO 8859-5	Mac Cyrillic	ISO 8859-5
215	275	332	257
216	276	333	377
217	277	334	360
220	300	335	241
221	301	336	361
222	302	337	357
223	303	340	320
224	304	341	321
225	305	342	322
226	306	343	323
227	307	344	324
230	310	345	325
231	311	346	326
232	312	347	327
233	313	350	330
234	314	351	331
235	315	352	332
236	316	353	333
237	317	354	334
240-246	40	355	335
247	246	356	336
250-252	40	357	337
253	242	360	340
254	362	361	341
255	40	362	342
256	243	363	343
257	363	364	344

Conversions Performed			
Mac Cyrillic	ISO 8859-5	Mac Cyrillic	ISO 8859-5
260-263	40	365	345
264	366	366	346
265-266	40	367	347
267	250	370	350
270	244	371	351
271	364	372	352
272	247	373	353
273	367	374	354
274	251	375	355
275	371	376	356
375	370		

Mac Cyrillic to KOI8-R

For the conversion of Mac Cyrillic to KOI8-R, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mac Cyrillic	KOI8-R	Mac Cyrillic	KOI8-R
24	4	276	272
200	341	277	252
201	342	300	250
202	367	301	265
203	347	302-311	40
204	344	312	240
205	345	313	261
206	366	314	241
207	372	315	274
210	351	316	254
211	352	317	245
212	353	320-327	40

Conversions Performed			
Mac Cyrillic	**KOI8-R**	**Mac Cyrillic**	**KOI8-R**
213	354	330	276
214	355	331	256
215	356	332	277
216	357	333	257
217	360	334	260
220	362	335	263
221	363	336	243
222	364	337	321
223	365	340	301
224	346	341	302
225	350	342	327
226	343	343	307
227	376	344	304
230	373	345	305
231	375	346	326
232	377	347	332
233	371	350	311
234	370	351	312
235	374	352	313
236	340	353	314
237	361	354	315
240-246	40	355	316
247	266	356	317
250-252	40	357	320
253	261	360	322
254	241	361	323
255	40	362	324

Conversions Performed			
Mac Cyrillic	**KOI8-R**	**Mac Cyrillic**	**KOI8-R**
256	262	363	325
257	242	364	306
260-263	40	365	310
264	246	366	303
265-266	40	367	336
267	270	370	333
270	264	371	335
271	244	372	337
272	267	373	331
273	247	374	330
274	271	375	334
275	251	376	300
375	370		

Mac Cyrillic to PC Cyrillic

For the conversion of Mac Cyrillic to PC Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mac Cyrillic	**PC Cyrillic**	**Mac Cyrillic**	**PC Cyrillic**
24	4	355	255
240-334	40	356	256
335	360	357	257
336	361	360	340
337	357	361	341
340	240	362	342
341	241	363	343
342	242	364	344
343	243	365	345
344	244	366	346

Conversions Performed			
Mac Cyrillic	PC Cyrillic	Mac Cyrillic	PC Cyrillic
345	245	367	347
346	246	370	350
347	247	371	351
350	250	372	352
351	251	373	353
352	252	374	354
353	253	375	355
354	254	376	356
303	366		

Mac Cyrillic to MS 1251

For the conversion of Mac Cyrillic to MS 1251, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mac Cyrillic	MS 1251	Mac Cyrillic	MS 1251
24	4	255	40
200	300	256	201
201	301	257	203
202	302	260-263	40
203	303	264	263
204	304	266	264
205	305	267	243
206	306	270	252
207	307	271	272
210	310	272	257
211	311	273	277
212	312	274	212
213	313	275	232
214	314	276	214

Conversions Performed			
Mac Cyrillic	**MS 1251**	**Mac Cyrillic**	**MS 1251**
215	315	277	234
216	316	300	274
217	317	301	275
220	320	302	254
221	321	303-306	40
222	322	307	253
223	323	310	273
224	324	311	205
225	325	312	240
226	326	313	200
227	327	314	220
230	330	315	215
231	331	316	235
232	332	317	276
233	333	320	226
234	334	321	227
235	335	322	223
236	336	323	224
237	337	324	221
240	206	325	222
241	260	326	40
242	245	327	204
243	40	330	241
244	247	331	242
245	267	332	217
246	266	333	237
247	262	334	271

Conversions Performed			
Mac Cyrillic	**MS 1251**	**Mac Cyrillic**	**MS 1251**
250	256	335	250
252	231	336	270
253	200	337	377
254	220	362	324

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_maz – code set conversion tables for Mazovia

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
Mazovia	maz	ISO 8859-2	iso2	ISO Latin 2
Mazovia	maz	MS 1250	win2	Windows Latin 2
Mazovia	maz	MS 852	dos2	MS-DOS Latin 2
Mazovia	maz	DHN	dhn	Dom Hanlowy Nauki

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

Mazovia to ISO 8859-2 For the conversion of Mazovia to ISO 8859-2, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mazovia	**ISO 8859-2**	**Mazovia**	**ISO 8859-2**
24–177	40	230	246
200	307	231	326
201	374	232	334
202	351	233	40
203	342	234	243
204	344	235	40
205	40	236	266
206	261	237	40
207	347	240	254
210	40	241	257
211	353	242	363
212-213	40	243	323
214	356	244	361
215	346	245	321

Conversions Performed			
Mazovia	ISO 8859-2	Mazovia	ISO 8859-2
216	304	246	274
217	241	247	277
220	312	250-340	40
221	352	341	337
222	263	342-365	40
223	364	366	367
224	366	367	40
225	306	370	260
226-227	40	371-376	40
256	201		

Mazovia to MS 1250

For the conversion of Mazovia to MS 1250, all characters not in the following table are mapped unchanged.

Mazovia	MS 1250	Mazovia	MS 1250
200	307	236	234
201	374	237	40
202	351	240	217
203	342	241	257
204	344	242	363
205	40	243	323
206	271	244	361
207	347	245	321
210	40	246	237
211	353	247	277
212-213	40	250-251	40
214	356	252	254
215	346	253-255	40
216	304	256	253

Mazovia	MS 1250	Mazovia	MS 1250
217	245	257	273
220	312	260-340	40
221	352	341	337
222	263	342-345	40
223	364	346	265
224	366	347-360	40
225	306	361	261
226-227	40	362-365	0
230	214	366	367
231	326	367	40
232	334	370	260
233	40	371	40
234	243	372	267
235	40	373-376	40
274	212		

Mazovia to MS 852

For the conversion of Mazovia to MS 852, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mazovia	MS 852	Mazovia	MS 852
205	40	234	235
206	245	235	40
210-213	40	236	230
215	206	237	40
217	244	240	215
220	250	241	275
221	251	243	340
222	210	244	344
225	217	245	343

Conversions Performed			
Mazovia	**MS 852**	**Mazovia**	**MS 852**
226-227	40	246	253
230	227	247	276
233	40	250-375	40
227	327		

Mazovia to DHN

For the conversion of Mazovia to DHN, all characters not in the following table are mapped unchanged.

Conversions Performed			
Mazovia	**DHN**	**Mazovia**	**DHN**
200-205	40	234	203
206	211	236	217
207-214	40	240	207
215	212	241	210
216	40	242	216
217	200	243	205
220	202	244	215
221	214	246	220
225	201	247	221
230	206		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_pc_cyr – code set conversion tables for Alternative PC Cyrillic

Description The following code set conversions are supported:

Code Set Conversions Supported				
Code	**Symbol**	**Target Code**	**Symbol**	**Target Output**
PC Cyrillic	alt	ISO 8859-5	iso5	ISO 8859-5 Cyrillic
PC Cyrillic	alt	KOI8-R	koi8	KOI8-R
PC Cyrillic	alt	MS 1251	win5	Windows Cyrillic
PC Cyrillic	alt	Mac Cyrillic	mac	Macintosh Cyrillic

Conversions The conversions are performed according to the following tables. All values in the tables are given in octal.

PC Cyrillic to ISO 8859-5 For the conversion of PC Cyrillic to ISO 8859-5, all characters not in the following table are mapped unchanged.

Conversions Performed			
PC Cyrillic	**ISO 8859-5**	**PC Cyrillic**	**ISO 8859-5**
24	4	231	311
200	260	232	312
201	261	233	313
202	262	234	314
203	263	235	315
204	264	236	316
205	265	237	317
206	266	240	320
207	267	241	321
210	270	242	322
211	271	243	323
212	272	244	324
213	273	245	325
214	274	246	326

Conversions Performed			
PC Cyrillic	ISO 8859-5	PC Cyrillic	ISO 8859-5
215	275	247	327
216	276	250	330
217	277	251	331
220	300	252	332
221	301	253	333
222	302	254	334
223	303	255	335
224	304	256	336
225	305	257	337
226	306	260-337	255
227	307	360	241
230	310	362-376	255

PC Cyrillic to KOI8-R

For the conversion of PC Cyrillic to KOI8-R, all characters not in the following table are mapped unchanged.

Conversions Performed			
PC Cyrillic	KOI8-R	PC Cyrillic	KOI8-R
24	4	242	327
200	341	243	307
201	342	244	304
202	367	245	305
203	347	246	326
204	344	247	332
205	345	250	311
206	366	251	312
207	372	252	313
210	351	253	314
211	352	254	315

Conversions Performed			
PC Cyrillic	KOI8-R	PC Cyrillic	KOI8-R
212	353	255	316
213	354	256	317
214	355	257	320
215	356	260-337	255
216	357	340	322
217	360	341	323
220	362	342	324
221	363	343	325
222	364	344	306
223	365	345	310
224	346	346	303
225	350	347	336
226	343	350	333
227	376	351	335
230	373	352	337
231	375	353	331
232	377	354	330
233	371	355	334
234	370	356	300
235	374	357	321
236	340	360	263
237	361	361	243
240	301	362-376	255
241	302		

PC Cyrillic to MS 1251

For the conversion of PC Cyrillic to MS 1251, all characters not in the following table are mapped unchanged.

Conversions Performed			
PC Cyrillic	**MS 1251**	**PC Cyrillic**	**MS 1251**
24	4	242	342
200	300	243	343
201	301	244	344
202	302	245	345
203	303	246	346
204	304	247	347
205	305	250	350
206	306	251	351
207	307	252	352
210	310	253	353
211	311	254	354
212	312	255	355
213	313	256	356
214	314	257	357
215	315	260-337	210
216	316	340	360
217	317	341	361
220	320	342	362
221	321	343	363
222	322	344	364
223	323	345	365
224	324	346	366
225	325	347	367
226	326	350	370
227	327	351	371
230	330	352	372
231	331	353	373

Conversions Performed			
PC Cyrillic	MS 1251	PC Cyrillic	MS 1251
232	332	354	374
233	333	355	375
234	334	356	376
235	335	357	377
236	336	360	250
237	337	361	270
240	340	362-376	210
241	341		

PC Cyrillic to Mac Cyrillic

For the conversion of PC Cyrillic to Mac Cyrillic, all characters not in the following table are mapped unchanged.

Conversions Performed			
PC Cyrillic	Mac Cyrillic	PC Cyrillic	Mac Cyrillic
24	4	341	361
240	340	342	362
241	341	343	363
242	342	344	364
243	343	345	365
244	344	346	366
245	345	347	367
246	346	350	370
247	347	351	371
250	350	352	372
251	351	353	373
252	352	354	374
253	353	355	375
254	354	356	376
255	355	357	337

Conversions Performed			
PC Cyrillic	**Mac Cyrillic**	**PC Cyrillic**	**Mac Cyrillic**
256	356	360	335
257	357	361	336
260-337	40	362-376	40
340	360		

Files

`/usr/lib/iconv/*.so`	conversion modules
`/usr/lib/iconv/*.t`	conversion tables
`/usr/lib/iconv/iconv_data`	list of conversions supported by conversion tables

See Also `iconv(1)`, `iconv(3C)`, `iconv(5)`

Name iconv_unicode – code set conversion tables for Unicode

Description The following code set conversions are supported:

CODE SET CONVERSIONS SUPPORTED

FROM Code Set Code	FROM Filename Element	TO Code Set Target Code	TO Filename Element
ISO 8859-1 (Latin 1)	8859-1	UTF-8	UTF-8
ISO 8859-2 (Latin 2)	8859-2	UTF-8	UTF-8
ISO 8859-3 (Latin 3)	8859-3	UTF-8	UTF-8
ISO 8859-4 (Latin 4)	8859-4	UTF-8	UTF-8
ISO 8859-5 (Cyrillic)	8859-5	UTF-8	UTF-8
ISO 8859-6 (Arabic)	8859-6	UTF-8	UTF-8
ISO 8859-7 (Greek)	8859-7	UTF-8	UTF-8
ISO 8859-8 (Hebrew)	8859-8	UTF-8	UTF-8
ISO 8859-9 (Latin 5)	8859-9	UTF-8	UTF-8
ISO 8859-10 (Latin 6)	8859-10	UTF-8	UTF-8
Japanese EUC	eucJP	UTF-8	UTF-8
Chinese/PRC EUC (GB 2312-1980)	gb2312	UTF-8	UTF-8
ISO-2022	iso2022	UTF-8	UTF-8
Korean EUC	ko_KR-euc	Korean UTF-8	ko_KR-UTF-8
ISO-2022-KR	ko_KR-iso2022-7	Korean UTF-8	ko_KR_UTF-8
Korean Johap (KS C 5601-1987)	ko_KR-johap	Korean UTF-8	ko_KR-UTF-8
Korean Johap (KS C 5601-1992)	ko_KR-johap92	Korean UTF-8	ko_KR-UTF-8
Korean UTF-8	ko_KR-UTF-8	Korean EUC	ko_KR-euc
Korean UTF-8	ko_KR-UTF-8	Korean Johap (KS C 5601-1987)	ko_KR-johap
Korean UTF-8	ko_KR-UTF-8	Korean Johap (KS C 5601-1992)	ko_KR-johap92
KOI8-R (Cyrillic)	KOI8-R	UCS-2	UCS-2
KOI8-R (Cyrillic)	KOI8-R	UTF-8	UTF-8
PC Kanji (SJIS)	PCK	UTF-8	UTF-8
PC Kanji (SJIS)	SJIS	UTF-8	UTF-8
UCS-2	UCS-2	KOI8-R (Cyrillic)	KOI8-R
UCS-2	UCS-2	UCS-4	UCS-4

CODE SET CONVERSIONS SUPPORTED

FROM Code Set Code	FROM Filename	TO Code Set Target Code	TO Filename

	Element		Element
UCS-2	UCS-2	UTF-7	UTF-7
UCS-2	UCS-2	UTF-8	UTF-8
UCS-4	UCS-4	UCS-2	UCS-2
UCS-4	UCS-4	UTF-16	UTF-16
UCS-4	UCS-4	UTF-7	UTF-7
UCS-4	UCS-4	UTF-8	UTF-8
UTF-16	UTF-16	UCS-4	UCS-4
UTF-16	UTF-16	UTF-8	UTF-8
UTF-7	UTF-7	UCS-2	UCS-2
UTF-7	UTF-7	UCS-4	UCS-4
UTF-7	UTF-7	UTF-8	UTF-8
UTF-8	UTF-8	ISO 8859-1 (Latin 1)	8859-1
UTF-8	UTF-8	ISO 8859-2 (Latin 2)	8859-2
UTF-8	UTF-8	ISO 8859-3 (Latin 3)	8859-3
UTF-8	UTF-8	ISO 8859-4 (Latin 4)	8859-4
UTF-8	UTF-8	ISO 8859-5 (Cyrillic)	8859-5
UTF-8	UTF-8	ISO 8859-6 (Arabic)	8859-6
UTF-8	UTF-8	ISO 8859-7 (Greek)	8859-7
UTF-8	UTF-8	ISO 8859-8 (Hebrew)	8859-8
UTF-8	UTF-8	ISO 8859-9 (Latin 5)	8859-9
UTF-8	UTF-8	ISO 8859-10 (Latin 6)	8859-10
UTF-8	UTF-8	Japanese EUC	eucJP
UTF-8	UTF-8	Chinese/PRC EUC (GB 2312-1980)	gb2312
UTF-8	UTF-8	ISO-2022	iso2022
UTF-8	UTF-8	KOI8-R (Cyrillic)	KOI8-R
UTF-8	UTF-8	PC Kanji (SJIS)	PCK
UTF-8	UTF-8	PC Kanji (SJIS)	SJIS
UTF-8	UTF-8	UCS-2	UCS-2
UTF-8	UTF-8	UCS-4	UCS-4
UTF-8	UTF-8	UTF-16	UTF-16
UTF-8	UTF-8	UTF-7	UTF-7
UTF-8	UTF-8	Chinese/PRC EUC (GB 2312-1980)	zh_CN.euc

CODE SET CONVERSIONS SUPPORTED

FROM Code Set Code	FROM Filename Element	TO Code Set Target Code	TO Filename Element
UTF-8	UTF-8	ISO 2022-CN	zh_CN.iso2022-7
UTF-8	UTF-8	Chinese/Taiwan Big5	zh_TW-big5
UTF-8	UTF-8	Chinese/Taiwan EUC (CNS 11643-1992)	zh_TW-euc

UTF-8	UTF-8	ISO 2022-TW	zh_TW-iso2022-7
Chinese/PRC EUC (GB 2312-1980)	zh_CN.euc	UTF-8	UTF-8
ISO 2022-CN	zh_CN.iso2022-7	UTF-8	UTF-8
Chinese/Taiwan Big5	zh_TW-big5	UTF-8	UTF-8
Chinese/Taiwan EUC (CNS 11643-1992)	zh_TW-euc	UTF-8	UTF-8
ISO 2022-TW	zh_TW-iso2022-7	UTF-8	UTF-8

Examples

EXAMPLE 1 The library module filename

In the conversion library, `/usr/lib/iconv` (see `iconv(3C)`), the library module filename is composed of two symbolic elements separated by the percent sign (`%`). The first symbol specifies the code set that is being converted; the second symbol specifies the *target code*, that is, the code set to which the first one is being converted.

In the conversion table above, the first symbol is termed the "FROM Filename Element". The second symbol, representing the target code set, is the "TO Filename Element".

For example, the library module filename to convert from the *Korean EUC* code set to the *Korean UTF-8* code set is

```
ko_KR-euc%ko_KR-UTF-8
```

Files

`/usr/lib/iconv/*.so` conversion modules

See Also

`iconv(1)`, `iconv(3C)`, `iconv(5)`

Chernov, A., *Registration of a Cyrillic Character Set*, RFC 1489, RELCOM Development Team, July 1993.

Chon, K., H. Je Park, and U. Choi, *Korean Character Encoding for Internet Messages*, RFC 1557, Solvit Chosun Media, December 1993.

Goldsmith, D., and M. Davis, *UTF-7 – A Mail-Safe Transformation Format of Unicode*, RFC 1642, Taligent, Inc., July 1994.

Lee, F., *HZ – A Data Format for Exchanging Files of Arbitrarily Mixed Chinese and ASCII characters*, RFC 1843, Stanford University, August 1995.

Murai, J., M. Crispin, and E. van der Poel, *Japanese Character Encoding for Internet Messages*, RFC 1468, Keio University, Panda Programming, June 1993.

Nussbacher, H., and Y. Bourvine, *Hebrew Character Encoding for Internet Messages*, RFC 1555, Israeli Inter-University, Hebrew University, December 1993.

Ohta, M., *Character Sets ISO-10646 and ISO-10646-J-1*, RFC 1815, Tokyo Institute of Technology, July 1995.

Ohta, M., and K. Handa, *ISO-2022-JP-2: Multilingual Extension of ISO-2022-JP*, RFC 1554, Tokyo Institute of Technology, December 1993.

Reynolds, J., and J. Postel, *ASSIGNED NUMBERS*, RFC 1700, University of Southern California/Information Sciences Institute, October 1994.

Simonson, K., *Character Mnemonics & Character Sets*, RFC 1345, Rationel Almen Planlaegning, June 1992.

Spinellis, D., *Greek Character Encoding for Electronic Mail Messages*, RFC 1947, SENA S.A., May 1996.

The Unicode Consortium, *The Unicode Standard*, Version 2.0, Addison Wesley Developers Press, July 1996.

Wei, Y., Y. Zhang, J. Li, J. Ding, and Y. Jiang, *ASCII Printable Characters-Based Chinese Character Encoding for Internet Messages*, RFC 1842, AsiaInfo Services Inc., Harvard University, Rice University, University of Maryland, August 1995.

Yergeau, F., *UTF-8, a transformation format of Unicode and ISO 10646*, RFC 2044, Alis Technologies, October 1996.

Zhu, H., D. Hu, Z. Wang, T. Kao, W. Chang, and M. Crispin, *Chinese Character Encoding for Internet Messages*, RFC 1922, Tsinghua University, China Information Technology Standardization Technical Committee (CITS), Institute for Information Industry (III), University of Washington, March 1996.

Notes ISO 8859 character sets using Latin alphabetic characters are distinguished as follows:

`ISO 8859-1 (Latin 1)` For most West European languages, including:

Albanian	Finnish	Italian
Catalan	French	Norwegian
Danish	German	Portuguese
Dutch	Galician	Spanish
English	Irish	Swedish
Faeroese	Icelandic	

`ISO 8859-2 (Latin 2)` For most Latin-written Slavic and Central European languages:

Czech	Polish	Slovak

	German	Rumanian	Slovene
	Hungarian	Croatian	

`ISO 8859-3 (Latin 3)`	Popularly used for Esperanto, Galician, Maltese, and Turkish.
`ISO 8859-4 (Latin 4)`	Introduces letters for Estonian, Latvian, and Lithuanian. It is an incomplete predecessor of ISO 8859-10 (Latin 6).
`ISO 8859-9 (Latin 5)`	Replaces the rarely needed Icelandic letters in ISO 8859-1 (Latin 1) with the Turkish ones.
`ISO 8859-10 (Latin 6)`	Adds the last Inuit (Greenlandic) and Sami (Lappish) letters that were not included in ISO 8859-4 (Latin 4) to complete coverage of the Nordic area.

Name ieee802.11 – 802.11 kernel statistics

Description This page describes the kernel statistics that can be used to monitor attributes specific to the 802.11 physical layer. These statistics can be retrieved using `kstat(1M)`. Not all 802.11 devices will support all statistics.

`tx_frags`	Count of data and management fragments transmitted.
`rx_frags`	Count of data and management fragments received.
`rx_dups`	Count of duplicate frames received. Duplicates are determined by the sequence control field.
`mcast_tx`	Count of broadcast and multicast frames transmitted.
`mcast_rx`	Count of broadcast and multicast frames received.
`tx_failed`	Count of frames that could not be transmitted due to the retransmission limit being reached.
`tx_retrans`	Count of frames successfully retransmitted after one or more retransmissions.
`tx_reretrans`	Count of frames successfully retransmitted after more than one retransmission.
`rts_success`	Count of times a CTS was received in response to an RTS.
`rts_failure`	Count of times a CTS was not received in response to an RTS.
`ack_failure`	Count of times an ACK was expected but was not received.
`fcs_errors`	Count of frames received with FCS errors.
`wep_errors`	Count of frames received with the WEP bit set but that either should not have been encrypted or that were discarded due to WEP not being supported.

See Also `kstat(1M)`

Name ieee802.3, cap_autoneg, cap_1000fdx, cap_1000hdx, cap_100fdx, cap_100hdx, cap_10fdx, cap_10hdx, cap_rem_fault, cap_pause, cap_asym_pause, adv_cap_autoneg, adv_cap_1000fdx, adv_cap_1000hdx, adv_cap_100fdx, adv_cap_100hdx, adv_cap_10fdx, adv_cap_10hdx, adv_cap_pause, adv_cap_asym_pause, adv_rem_fault, lp_cap_autoneg, lp_cap_1000fdx, lp_cap_1000hdx, lp_cap_100fdx, lp_cap_100hdx, lp_cap_10fdx, lp_cap_10hdx, lp_cap_pause, lp_cap_asym_pause, lp_rem_fault, xcvr_addr, xcvr_id, xcvr_inuse, link_up, link_duplex, link_tx_pause, link_rx_pause – Ethernet mii kstat and dladm parameters

Description This page describes the kernel statistics and the `dladm(1M)` configuration parameters used to monitor and configure the Ethernet physical layer.

The `cap_*` parameters exist in the kernel statistics for an Ethernet device. The parameters describe the maximum capability of a device. When the value of a statistic is `1`, the device has the capability described. When the value is `0`, the device does not have the capability.

The exceptions to this rule are the `cap_asym_pause` and `cap_pause` parameters which are explained later in this page.

`cap_autoneg`	Capable of auto-negotiation
`cap_1000fdx`	Capable of 1000 full duplex operation
`cap_1000hdx`	Capable of 1000 half duplex operation
`cap_100fdx`	Capable of 100 full duplex operation
`cap_100hdx`	Capable of 100 half duplex operation
`cap_10fdx`	Capable of 10 full duplex operation
`cap_10hdx`	Capable of 10 half duplex operation
`cap_rem_fault`	Capable of reporting locally detected faults to link partner

The `adv_cap_*` parameters exist in the kernel statistics and represent a mirror image of the dladm adv_*_cap parameter list for an Ethernet device. The dladm adv_*_cap tuning parameters allow fine grain control of the Ethernet device physical layer. The parameters are also a subset of the `cap_*` statistics. If the `cap_*` value is `0`, the corresponding `adv_cap_*` must also be `0`. The exceptions to this rule are the `adv_cap_asym_pause` and `adv_cap_pause` parameters.

When auto-negotiation is enabled, the `adv_*_cap` statistics show which capabilities are advertised to the link partner. When auto-negotiation is disabled in *forced mode*, the statistics precisely show how a link should function and that it must be matched on the link partner to achieve a valid link up.

Statistics with values other than `0` and `1` are also described in the following.

`adv_cap_autoneg` Advertise auto-negotiation capability

`adv_cap_1000fdx` Advertise 1000 full duplex capability

`adv_cap_1000hdx` Advertise 1000 half duplex capability

`adv_cap_100fdx` Advertise 100 full duplex capability

`adv_cap_100hdx` Advertise 100 half duplex capability

`adv_cap_10fdx` Advertise 10 full duplex capability

`adv_cap_10hdx` Advertise 10 half duplex capability

`adv_rem_fault` Fault value reported by the local system to the peer

- `0` Link is good
- `1` Fault

The `lp_cap_*` parameters exist as kernel statistics for an Ethernet device. The statistics are the advertised capabilities provided by the link partner on completion of auto-negotiation. If the capabilities match the capabilities provided in the local advertisement, the link can proceed to a link up state. If no match is found, the link remains down. In two other instances, `lp_cap_*` values might all be zero: (1) when a cable is not present and (2) when forced mode is enabled.

`lp_cap_autoneg` Link partner advertises auto-negotiation capability

`lp_cap_1000fdx` Link partner advertises 1000 full duplex capability

`lp_cap_1000hdx` Link partner advertises 1000 half duplex capability

`lp_cap_100fdx` Link partner advertises 100 full duplex capability

`lp_cap_100hdx` Link partner advertises 100 half duplex capability

`lp_cap_10fdx` Link partner advertises 10 full duplex capability

`lp_cap_10hdx` Link partner advertises 10 half duplex capability

`lp_rem_fault` Fault value the remote system reports

- `0` Link is good
- `1` Fault

The `xcvr_*` kernel statistics provide information about the physical layer device that is in use.

`xcvr_addr` MII address in the `0` to `31` range of the physical layer device in use for a given Ethernet device

`xcvr_id` MII transceiver manufacturer and device ID

`xcvr_inuse` MII transceiver type, based on the following list:

0 other	Undefined
1 none	MII present, but nothing connected
2 10Mb/s	10Mb/s Manchester encoding
3 100BaseT4	100 Mb/s 8B/6T
4 100BaseX	100 Mb/s 4B/5B
5 100BaseT2	100 Mb/s PAM5X5
6 1000BaseX	1000 Mb/s 8B/10B
7 1000BaseT	1000 Mb/s 4D-PAM5

The above values define maximum capability. In many cases, lower speeds can occur. The `cap_*` statistics must be viewed to establish the range of capability.

The `link_*` kernel statistics show the link state at the local end of the connection.

`link_up`	1	Link is up
	0	Link is down
`link_duplex`	2	Full duplex link
	1	Half duplex link
	0	Unknown

The `cap_asym_pause`, `cap_pause`, `adv_cap_asym_pause`, and `adv_cap_pause` parameters do not follow the rules of other `cap_*` and `adv_cap_*` kstats or parameters. The `cap_*pause` kstats provide information about the capabilities supported by the device and constrain the values that may be set to the corresponding `adv_cap_*pause` parameters.

`cap_pause` Symmetric pause capability.

`cap_asym_pause` Asymmetric pause capability.

The `adv_cap_pause` and `adv_cap_asym_pause` statistics are limited by the available settings for `cap_pause` and `cap_asym_pause`. These statistics are read-only values whose settings may be administratively controlled by setting the flowctrl property supported by `dladm(1M)`. For a device that is fully capable of pausing both Rx (receive) and Tx (transmit) operations, the settings available are defined in the truth table that follows the `adv_cap_pause` and `adv_cap_asm_pause` parameter descriptions below.

`adv_cap_pause` When adv_cap_pause is 1, the device can both assert and respond to flow control. This is the pre-Gigabit, symmetric mode of

operation, and implies a full (both send and receive) implementation of the PAUSE mechanism within the device. In addition, if adv_cap_asym_pause is 1, the device can operate either symmetrically or asymmetrically in either direction.

If adv_cap_pause is 0, advertised, flow-control behavior is determined by adv_cap_asym_pause. If the value of adv_cap_asym_pause is 1, the device can assert flow control, but cannot resend.

No flow control is available when bot adv_cap_pause and adv_cap_asym_pause are 0.

`adv_cap_asym_pause` Asymmetric pause capability.

The `cap_asym_pause` and `cap_pause` statistics show the capability of a device and also limit the legal setting for `adv_cap_asym_pause` and `adv_cap_pause`. The following truth table describes the available `adv_cap_asym_pause` and `adv_cap_pause` settings limited by `cap_asym_pause` and `cap_pause` `statistics`. The abbreviations below are used in the table.

CA `cap_asym_pause`

CP `cap_pause`

AA `adv_cap_asym_pause`

AP `adv_cap_pause`

CP	CA	AP	AA	Description
0	0	0	0	No pause in use.
0	0	x	x	Device not pause capable, cannot set.
0	1	0	0	Asymmetric Rx pause capable, but not advertised.
0	1	0	1	Asymmetric Rx pause capable and advertised.
0	1	1	0	Asymmetric Rx pause capable, but not advertised. Not capable of symmetric pause.
0	1	1	1	Asymmetric Rx pause capable and advertised. No symmetric pause capability or asymmetric Tx pause.
1	0	0	0	Symmetric pause capable, but not advertised.
1	0	0	1	Symmetric pause capable, advertising asymmetric Rx pause only.
1	0	1	0	Symmetric pause capable, advertising symmetric Rx and Tx pause capability.

1	0	1	1	Symmetric pause capable and advertised.
1	1	0	0	Symmetric and asymmetric pause capable, but not advertised.
1	1	0	1	Symmetric and asymmetric Tx pause capable. Only asymmetric Tx pause advertised.
1	1	1	0	Symmetric and symmetric Tx pause capable. Only symmetric pause advertised.
1	1	1	1	Asymmetric Tx pause capable and advertised.

In the cases above, an error is posted when a device driver cannot advertise. A new setting is ignored and values revert to the previous setting.

The `lp_cap_pause` and the `lp_cap_asym_pause` provide the advertised capabilities of the link partners.

`lp_cap_pause` When `lp_cap_pause` is 1, the link-partner can both assert and respond to flow control. This is the pre-Gigabit, symmetric mode of operation, and implies a full (both send and receive) implementation of the PAUSE mechanism within the device. In addition, if `lp_cap_asym_pause` is 1, the link-partner can operate either symmetrically or asymmetrically in either direction.

If `lp_cap_pause` is 0, the flow-control behavior supported by the link-partner is determined by `lp_cap_asym_pause`. If the value of `lp_cap_asym_pause` is 1, the link-partner can assert flow control, but cannot respond to any pause-frames sent to it.

No flow control is available when both `lp_cap_pause` and `lp_cap_asym_pause` are 0.

`lp_cap_asym_pause` Asymmetric pause capability

When `adv_*pause_cap` and `lp_*pause_cap` are compared on completion of auto-negotiation, the chosen flow control mechanism for the link depends on what is most meaningful.

`link_tx_pause` Link partner can assert flow control by sending pause frames when congestion is experienced.

`link_rx_pause` Link partner can respond to pause frames received.

The following truth table illustrates the meaningful flow control combinations related to local and link partner configurations. The abbreviations below are used in the table.

AA `adv_cap_asym_pause`

AP `adv_cap_pause`

LAC `lp_cap_asym_pause`

LPC `lp_cap_pause`

LA `link_asym_pause`

LP `link_pause`

AA	AP	LAC	LPC	LA	LP	Description
1	0	1	1	1	0	Local station will Tx a pause when Rx is congested.
0	1	0	1	0	1	Flow control in both Rx and Tx directions.
x	1	1	0	1	1	Local station honors received Pause frames by temporarily suspending Transmit.
x	x	x	x	0	0	All other combinations: Flow control not available on the link

When forced mode is enabled, the current setting of `adv_cap_asym_pause` and `adv_cap_pause` are used for the link. The `link_asym_pause` and `link_pause` become equal to the current `adv_cap_asym_pause` and `adv_cap_pause` settings. The above table also applies in forced mode, but the link partner configuration must be checked to verify that flow control is operating on the link.

See Also `dladm`(1M), `driver.conf`(4), `bge`(7D), `ce`(7D), `dlpi`(7P), `eri`(7D), `ge`(7D), `gld`(7D), `hme`(7D), `qfe`(7d)

Notes When `adv_cap_autoneg` is set to `0`, the highest priority speed and duplex is used for forced mode.

The highest priority is the highest speed at full duplex. The lowest priority is the lowest speed at half duplex.

MII transceivers can exist internally to a system or can be connected to an external MII connector. Typically, an internal transceiver has an `xcvr_addr` of `1`, while an external connection has an `xcvr_addr` of `0`.

Name ipfilter – IP packet filtering software

Description IP Filter is software that provides packet filtering capabilities on a Solaris system. On a properly setup system, it can be used to build a firewall.

Solaris IP Filter is installed with the Solaris operating system. However, packet filtering is not enabled by default. See `ipf(1M)` for a procedure to enable and activate the IP Filter feature.

Host-Based Firewall To simplify IP Filter configuration management, a firewall framework is created to allow users to configure IP Filter by expressing firewall policy at system and service level. Given the user-defined firewall policy, the framework generates a set of IP Filter rules to enforce the desired system behavior. Users specify system and service firewall policies that allow or deny network traffic from certain hosts, subnets, and interface(s). The policies are translated into a set of active IPF rules to enforce the specified firewall policies.

Note – Users can still specify their own ipf rule file if they choose not to take advantage of the framework. See `ipf(1M)` and `ipf(4)`.

Model This section describes the host-based firewall framework. See svc.ipfd(1M) for details on how to configure firewall policies.

A three-layer approach with different precedence levels helps the user achieve the desired behaviors.

Global Default
: Global Default - Default system-wide firewall policy. This policy is automatically inherited by all services unless services modify their firewall policy.

Network Services
: Higher precedence than Global Default. A service's policy allows/disallows traffic to its specific ports, regardless of Global Default policy.

Global Override
: Another system-wide policy that takes precedence over the needs of specific services in Network Services layer.

```
Global Override
      |
      |
Network Services
      |
      |
Global Default
```

A firewall policy includes a firewall mode and an optional set of network sources. Network sources are IP addresses, subnets, and local network interfaces, from all of which a system can receive incoming traffic. The basic set of firewall modes are:

None
: No firewall, allow all incoming traffic.

Deny
Allow all incoming traffic but deny from specified source(s).

Allow
Deny all incoming traffic but allow from specified source(s).

Layers in Detail

The first system-wide layer, Global Default, defines a firewall policy that applies to *any* incoming traffic, for example, allowing or blocking all traffic from an IP address. This makes it simple to have a policy that blocks all incoming traffic or all incoming traffic from unwanted source(s).

The Network Services layer contains firewall policies for local programs that provide service to remote clients, for example, `telnetd`, `sshd`, and `httpd`. Each of these programs, a network service, has its own firewall policy that controls access to its service. Initially, a service's policy is set to inherit Global Default policy, a "Use Global Default" mode. This makes it simple to set a single policy, at the Global Default layer, that can be inherited by all services.

When a service's policy is different from Global Default policy, the service's policy has higher precedence. If Global Default policy is set to block all traffic from a subnet, the SSH service could be configured to allow access from certain hosts in that subnet. The set of all policies for all network services comprises the Network Service layer.

The second system-wide layer, Global Override, has a firewall policy that also applies to any incoming network traffic. This policy has highest precedence and overrides policies in the other layers, specifically overriding the needs of network services. The example is when it is desirable to block known malicious source(s) regardless of services' policies.

User Interaction

This framework leverages IP Filter functionality and is active only when `svc:/network/ipfilter` is enabled and inactive when `network/ipfilter` is disabled. Similarly, a network service's firewall policy is only active when that service is enabled and inactive when the service is disabled. A system with an active firewall has IP Filter rules for each running/enabled network service and system-wide policy(s) whose firewall mode is not `None`.

A user configures a firewall by setting the system-wide policies and policy for each network service. See svc.ipfd(1M) on how to configure a firewall policy.

The firewall framework composes of policy configuration and a mechanism to generate IP Filter rules from the policy and applying those rules to get the desired IP Filter configuration. A quick summary of the design and user interaction:

- system-wide policy(s) are stored in `network/ipfilter`
- network services' policies are stored in each SMF service
- a user activates a firewall by enabling `network/ipfilter` (see `ipf(1M)`)
- a user activates/deactivate a service's firewall by enabling/disabling that network service
- changes to system-wide or per-service firewall policy results in an update to the system's firewall rules

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

See Also svcs(1), ipf(1M), ipnat(1M), svcadm(1M), svc.ipfd(1M), ipf(4), ipnat(4), attributes(5), smf(5)

System Administration Guide: IP Services

Notes The ipfilter service is managed by the service management facility, smf(5), under the service identifier:

```
svc:/network/ipfilter:default
```

Administrative actions on this service, such as enabling, disabling, or requesting restart, can be performed using svcadm(1M). The service's status can be queried using the svcs(1) command.

IP Filter startup configuration files are stored in /etc/ipf.

Name ipkg – ipkg branded zone

Description The `ipkg` brand uses the branded zones framework described in `brands(5)` to run zones installed with the same software as is installed in the global zone. The system software must always be in sync with the global zone when using a `ipkg` brand. The system software packages within the zone are managed using the image packaging system. See `pkg(5)`.

Configuration and Administration

The `ipkg` brand supports the whole root non-global zone model. All of the required system software and any additional packages are installed into the private file systems of the zone. The zone must reside on its own `zfs(1M)` dataset and only ZFS is supported. The ZFS dataset is created automatically when the zone is installed or attached. If a ZFS dataset cannot be created, the zone is not installed or attached.

Sub-commands The following `ipkg` brand-specific subcommand options are supported by `zoneadm(1M)`.

`attach` [`-a` *archive* | `-d` *path*] [`-u`]

Attach the specified `ipkg` branded zone image into the zone. If neither `-a` or `-d` is specified, the zone's zonepath is assumed to already be properly installed with the zone's files. `zoneadm` checks package levels on the machine to which the zone is to be attached. If the packages that the zone depends on from the global zone are different (have different revision numbers) from the dependent packages on the source machine, `zoneadm` reports these conflicts and does not perform the attach. If the destination system has only newer dependent packages (higher revision numbers) than those on the source system, you can use the `-u`option to update the attached zone to match the revision of the packages that exist on the new system. With `-u`, as in the default behavior, `zoneadm` does not perform an attach if outdated packages are found on the target system.

`-a` *archive*

The path to a `cpio(1)` or `pax(1)` `xustar` archive of an installed `ipkg` branded zone's zonepath. `cpio` archives can be compressed using `gzip` or `bzip2`.

`-d` *path*

The path to the zonepath directory of a `ipkg` branded zone's zonepath.

`-u`

Update the packages for the zone's system software to match the global zone.

`install` [`-c` *certificate_file*] [`-k` *key_file*] [`-P` *publisher=uri*] [`-e` *extrapkg* [...]]
`install` [`-a` *archive*] | [`-d` *path*] [`-p`] [`-s`] [`-u`] [`-v`]

The `ipkg` brand installer supports installing the zone from either the software repository or from an image of an installed system running the same release. This can be a `cpio(1)` or `pax(1)` `xustar` archive. The `cpio` archive can be compressed with `gzip` or `bzip2`. The image can also be a path to the top-level of a system's root tree, or a pre-existing zone path.

If neither the `-a` or `-d` option is used, the zone is installed from the repository.

To install the zone from a system image either the `-a` or `-d` is required. Either the `-u` or `-p` option is also required in this case.

`-a` *archive*	The path to a `cpio(1)` or `pax(1)` `xustar`archive of an installed system. `cpio` archives may be compressed using `gzip` or `bzip2`.
`-c` *certificate_file*	The path to the certificate file needed for accessing the repository.
`-d` *path*	The path to the root directory of an installed system. If path is a hyphen (`-`), the zonepath is presumed to be already populated with the system image
`-e` *package_name*	The name of a package to install into the zone. The `-e` option is used to specify an additional package to install, in addition to the base set of software which is always installed into the zone. Multiple `-e` options can be used
`-k` *key_file*	The path to the key file needed for accessing the repository.
`-P` *publisher=uri*	The name of a software repository publisher and its associated URI to use to install the zone instead of the default global zone's repository.
`-p`	Preserve the system configuration after installing the zone.
`-s`	Install silently
`-u`	Run `sys-unconfig(1M)` on the zone after installing it.
`-v`	Verbose output from the install process.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWzoneu
Interface Stability	Uncommitted

See Also `cpio(1)`, `pax(1)`, `zfs(1M)`, `zlogin(1)`, `zonename(1)`, `sys-unconfig(1M)`, `zoneadm(1M)`, `zonecfg(1M)`,, `attributes(5)`, `brands(5)`, `privileges(5)`, `zones(5)`

`pkg(5)`, available in the IPS consolidation

Name isalist – the native instruction sets known to Solaris software

Description The possible instruction set names returned by `isalist(1)` and the `SI_ISALIST` command of `sysinfo(2)` are listed here.

The list is ordered within an instruction set family in the sense that later names are generally faster then earlier names; note that this is in the reverse order than listed by `isalist(1)` and `sysinfo(2)`. In the following list of values, numbered entries generally represent increasing performance; lettered entries are either mutually exclusive or cannot be ordered.

This feature is obsolete and may be removed in a future version of Solaris. The lists below do not reflect all the extensions that have been made by modern processors. See `getisax(2)` for a better way to handle instruction set extensions.

SPARC Platforms Where appropriate, correspondence with a given value of the `-xarch` option of Sun's C 4.0 compiler is indicated. Other compilers might have similar options.

1a. `sparc`	Indicates the SPARC V8 instruction set, as defined in The SPARC Architecture Manual, Version 8, Prentice-Hall, Inc., 1992. Some instructions (such as integer multiply and divide, FSMULD, and all floating point operations on quad operands) can be emulated by the kernel on certain systems.
1b. `sparcv7`	Same as sparc. This corresponds to code produced with the –xarch=v7 option of Sun's C 4.0 compiler.
2. `sparcv8–fsmuld`	Like sparc, except that integer multiply and divide must be executed in hardware. This corresponds to code produced with the –xarch=v8a option of Sun's C 4.0 compiler.
3. `sparcv8`	Like sparcv8–fsmuld, except that FSMULD must also be executed in hardware. This corresponds to code produced with the –xarch=v8 option of Sun's C 4.0 compiler.
4. `sparcv8plus`	Indicates the SPARC V8 instruction set plus those instructions in the SPARC V9 instruction set, as defined in The SPARC Architecture Manual, Version 9, Prentice-Hall, 1994, that can be used according to The V8+ Technical Specification. This corresponds to code produced with the –xarch=v8plus option of Sun's C 4.0 compiler.
5a. `sparcv8plus+vis`	Like sparcv8plus, with the addition of those UltraSPARC I Visualization Instructions that can be used according to The V8+ Technical Specification. This corresponds to code produced with the –xarch=v8plusa option of Sun's C 4.0 compiler.
5b. `sparcv8plus+fmuladd`	Like sparcv8plus, with the addition of the Fujitsu SPARC64 floating multiply-add and multiply-subtract instructions.

	6. `sparcv9`	Indicates the SPARC V9 instruction set, as defined in The SPARC Architecture Manual, Version 9, Prentice-Hall, 1994.
	7a. `sparcv9+vis`	Like sparcv9, with the addition of the UltraSPARC I Visualization Instructions.
	7b. `sparcv9+vis2`	Like sparcv9, with the addition of the UltraSPARC III Visualization Instructions.
	7c. `sparcv9+fmuladd`	Like sparcv9, with the addition of the Fujitsu SPARC64 floating multiply-add and multiply-subtract instructions.
x86 Platforms	1. `i386`	The Intel 80386 instruction set, as described in The i386 Microprocessor Programmer's Reference Manual.
	2. `i486`	The Intel 80486 instruction set, as described in The i486 Microprocessor Programmer's Reference Manual. (This is effectively i386, plus the CMPXCHG, BSWAP, and XADD instructions.)
	3. `pentium`	The Intel Pentium instruction set, as described in The Pentium Processor User's Manual. (This is effectively i486, plus the CPU_ID instruction, and any features that the CPU_ID instruction indicates are present.)
	4. `pentium+mmx`	Like pentium, with the MMX instructions guaranteed present.
	5. `pentium_pro`	The Intel PentiumPro instruction set, as described in The PentiumPro Family Developer's Manual. (This is effectively pentium, with the CMOVcc, FCMOVcc, FCOMI, and RDPMC instructions guaranteed present.)
	6. `pentium_pro+mmx`	Like pentium_pro, with the MMX instructions guaranteed present.
	7. `amd64`	The AMD Opteron instruction set, as described in the *AMD64 Architecture Programmer's Manual*.

See Also `isalist(1)`, `getisax(2)`, `sysinfo(2)`

Name kerberos – overview of Solaris Kerberos implementation

Description The Solaris Kerberos implementation, hereafter sometimes shortened to "Kerberos," authenticates clients in a network environment, allowing for secure transactions. (A client may be a user or a network service.) Kerberos validates the identity of a client and the authenticity of transferred data. Kerberos is a *single-sign-on* system, meaning that a user needs to provide a password only at the beginning of a session. The Solaris Kerberos implementation is based on the Kerberos(TM) system developed at MIT, and is compatible with Kerberos V5 systems over heterogeneous networks.

Kerberos works by granting clients *tickets*, which uniquely identify a client, and which have a finite lifetime. A client possessing a ticket is automatically validated for network services for which it is entitled; for example, a user with a valid Kerberos ticket may rlogin into another machine running Kerberos without having to identify itself. Because each client has a unique ticket, its identity is guaranteed.

To obtain tickets, a client must first initialize the Kerberos session, either by using the `kinit(1)` command or a PAM module. (See `pam_krb5(5)`). `kinit` prompts for a password, and then communicates with a *Key Distribution Center* (KDC). The KDC returns a *Ticket-Granting Ticket* (TGT) and prompts for a confirmation password. If the client confirms the password, it can use the Ticket-Granting Ticket to obtain tickets for specific network services. Because tickets are granted transparently, the user need not worry about their management. Current tickets may be viewed by using the `klist(1)` command.

Tickets are valid according to the system *policy* set up at installation time. For example, tickets have a default lifetime for which they are valid. A policy may further dictate that privileged tickets, such as those belonging to root, have very short lifetimes. Policies may allow some defaults to be overruled; for example, a client may request a ticket with a lifetime greater or less than the default.

Tickets can be renewed using `kinit`. Tickets are also *forwardable*, allowing you to use a ticket granted on one machine on a different host. Tickets can be destroyed by using `kdestroy(1)`. It is a good idea to include a call to `kdestroy` in your `.logout` file.

Under Kerberos, a client is referred to as a *principal*. A principal takes the following form:

```
primary/instance@REALM
```

primary — A user, a host, or a service.

instance — A qualification of the primary. If the primary is a host — indicated by the keyword `host`— then the instance is the fully-qualified domain name of that host. If the primary is a user or service, then the instance is optional. Some instances, such as `admin` or `root`, are privileged.

realm — The Kerberos equivalent of a domain; in fact, in most cases the realm is directly mapped to a DNS domain name. Kerberos realms are given in upper-case only.

For examples of principal names, see the EXAMPLES.

By taking advantage of the General Security Services API (GSS-API), Kerberos offers, besides user authentication, two other types of security service: *integrity*, which authenticates the validity of transmitted data, and *privacy*, which encrypts transmitted data. Developers can take advantage of the GSS-API through the use of the RPCSEC_GSS API interface (see `rpcsec_gss(3NSL)`).

Examples

EXAMPLE 1 Examples of valid principal names

The following are examples of valid principal names:

```
joe
joe/admin
joe@ENG.ACME.COM
joe/admin@ENG.ACME.COM
rlogin/bigmachine.eng.acme.com@ENG.ACME.COM
host/bigmachine.eng.acme.com@ENG.ACME.COM
```

The first four cases are *user principals*. In the first two cases, it is assumed that the user `joe` is in the same realm as the client, so no realm is specified. Note that `joe`and `joe/admin` are different principals, even if the same user uses them; `joe/admin` has different privileges from `joe`. The fifth case is a *service principal*, while the final case is a *host principal*. The word `host` is required for host principals. With host principals, the instance is the fully qualified hostname. Note that the words `admin` and `host` are reserved keywords.

See Also

`kdestroy(1)`, `kinit(1)`, `klist(1)`, `kpasswd(1)`, `krb5.conf(4)`, `krb5envvar(5)`

System Administration Guide: Security Services

Notes

In previous releases of the Solaris operating system, the Solaris Kerberos implementation was referred to as the "Sun Enterprise Authentication Mechanism" (SEAM).

If you enter your username and `kinit` responds with this message:

```
Principal unknown (kerberos)
```

you have not been registered as a Kerberos user. See your system administrator or the *System Administration Guide: Security Services*.

Name krb5_auth_rules – overview of Kerberos V5 authorization

Description When kerberized versions of the `ftp`, `rdist`, `rcp`, `rlogin`, `rsh`, `telnet`, or `ssh` clients are used to connect to a server, the identity of the originating user must be authenticated to the Kerberos V5 authentication system. Account access can then be authorized if appropriate entries exist in the `~/.k5login` file, the `gsscred` table, or if the default GSS/Kerberos authentication rules successfully map the Kerberos principal name to Unix login name.

To avoid security problems, the `~/.k5login` file must be owned by the remote user on the server the client is attempting to access. The file should contain a private authorization list comprised of Kerberos principal names of the form *principal/instance@realm*. The */instance* variable is optional in Kerberos principal names. For example, different principal names such as `jdb@ENG.ACME.COM` and `jdb/happy.eng.acme.com@ENG.ACME.COM` would each be legal, though not equivalent, Kerberos principals. The client is granted access if the `~/.k5login` file is located in the login directory of the remote user account and if the originating user can be authenticated to one of the principals named in the file. See `gkadmin(1M)` and `kadm5.acl(4)` for more information on Kerberos principal names.

When no `~/.k5login` file is found in the remote user's login account, the Kerberos V5 principal name associated with the originating user is checked against the `gsscred` table. If a `gsscred` table exists and the principal name is matched in the table, access is granted if the Unix user ID listed in the table corresponds to the user account the client is attempting to access. If the Unix user ID does not match, access is denied. See `gsscred(1M)`.

For example, an originating user listed in the `gsscred` table with the principal name `jdb@ENG.ACME.COM` and the `uid` `23154` is granted access to the `jdb-user` account if `23154` is also the `uid` of `jdb-user` listed in the user account database. See `passwd(4)`.

Finally, if there is no `~/.k5login` file and the Kerberos V5 identity of the originating user is not in the `gsscred` table, or if the `gsscred` table does not exist, the client is granted access to the account under the following conditions (default GSS/Kerberos auth rules):

- The user part of the authenticated principal name is the same as the Unix account name specified by the client.
- The realm part of the client and server are the same, unless the `krb5.conf(4)` *auth_to_local_realm* parameter is used to create equivalence.
- The Unix account name exists on the server.

For example, if the originating user has the principal name `jdb@ENG.ACME.COM` and if the server is in realm `SALES.ACME.COM`, the client would be denied access even if `jdb` is a valid account name on the server. This is because the realms `SALES.ACME.COM` and `ENG.ACME.COM` differ.

The krb5.conf(4) *auth_to_local_realm* parameter also affects authorization. Non-default realms can be equated with the default realm for authenticated `name-to-local name` mapping.

Files

`~/.k5login`	Per user-account authorization file.
`/etc/passwd`	System account file. This information may also be in a directory service. See passwd(4).

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also ftp(1), rcp(1), rdist(1), rlogin(1), rsh(1), telnet(1), gkadmin(1M), gsscred(1M), kadm5.acl(4), krb5.conf(4), passwd(4), attributes(5), gss_auth_rules(5)

Name krb5envvar – Kerberos environment variables

Description The Kerberos mechanism provides a number of environment variables to configure different behavior in order to meet applications' needs. Environment variables used within the Kerberos mechanism are:

`KRB5_KTNAME`
Used by the mechanism to specify the location of the key table file. The variable can be set to the following value:

`[[`<*kt type*>`:]`<*file name*>`]`

where <*kt type*> can be `FILE` or `WRFILE`. `FILE` is for read operations; `WRFILE` is for write operations. <*file name*> is the location of the `keytab` file.

r

If `KRB5_KTNAME` is not defined, the default value is:

`FILE:/etc/krb5/krb5.keytab`

The `keytab` file is used to store credentials persistently and is used commonly for service daemons.

Specifying the `FILE` type assumes that the subsequent operations on the associated file are readable by the invoking process. Care must be taken to ensure that the file is readable only by the set of principals that need to retrieve their unencrypted keys.

The `WRFILE` type is used by the `kadmin(1M)` command. Specifying this type allows the administrator to designate an alternate `keytab` file to write to without using extra command line arguments for file location.

`KRB5CCNAME`
Used by the mechanism to specify the location of the credential cache. The variable can be set to the following value:

`[[`<*cc type*>`:]`<*file name*>`]`

where <*cc type*> can be `FILE` or `MEMORY`. <*file name*> is the location of the principal's credential cache.

If `KRB5CCNAME` is not defined, the default value is:

`FILE:/tmp/krb5cc_`<*uid*>

where <*uid*> is the user id of the process that created the cache file.

The credential cache file is used to store tickets that have been granted to the principal.

Specifying the `FILE` types assumes that subsequent operations on the associated file are readable and writable by the invoking process. Care must be taken to ensure that the file is

accessible only by the set of principals that need to access their credentials. If the credential file is in a directory to which other users have write access, you need to set that directory's sticky bit (see chmod(1)).

The MEMORY credential cache type is used only in special cases, such as when making a temporary cache for the life of the invoking process.

KRB5RCNAME

Used by the mechanism to specify the type and location of the replay cache. The variable can be set to the following value:

[[<*rc type*>:]<*file name*>]

where <*rc type*> can be either FILE, MEMORY, or NONE. <*file name*> is relevant only when specifying the replay cache file type.

If not defined, the default value is:

FILE:/var/krb5/rcache/root/rc_<*service*>

...if the process is owned by root, or:

FILE:/var/krb5/rcache/rc_<*service*>

...if the process is owned by a user other than root. <*service*> is the service process name associated with the replay cache file.

The replay cache is used by Kerberos to detect the replay of authentication data. This prevents people who capture authentication messages on the network from authenticating to the server by resending these messages.

When specifying the FILE replay cache type, care must be taken to prevent the replay cache file from being deleted by another user. Make sure that every directory in the replay cache path is either writable only by the owner of the replay cache or that the sticky bit ("t") is set on every directory in the replay cache path to which others have write permission.

When specifying the MEMORY replay cache type you need to weigh the trade-off of performance against the slight security risk created by using a non-persistent cache. The risk occurs during system reboots when the following condition obtains:

- The duration from the last write to the replay cache before reboot to the point when the Kerberized server applications are running is less than the Kerberos clockskew (see krb5.conf(4)).

When specifying the NONE replay cache time you need to understand that this disables the replay cache, and all security risks that this presents. This includes all the risks outlined in this section of the man page.

Under this condition, the server applications can accept a replay of Kerberos authentication data (up to the difference between the time of the last write and the clockskew). Typically, this is a small window of time. If the server applications take longer than the clockskew to start accepting connections there is no replay risk.

The risk described above is the same when using `FILE` replay cache types when the replay cache resides on swap file systems, such as `/tmp` and `/var/run`.

The performance improvement in `MEMORY` replay cache types over `FILE` types is derived from the absence of disk I/O. This is true even if the `FILE` replay cache is on a memory-backed file system, such as swap (`/tmp` and `/var/run`).

Note that `MEMORY`-type caches are per-process caches, therefore use of these types of caches must be carefully considered. One example of where `MEMORY`-type caches can be problematic is when an application uses more than one process for establishing security contexts. In such a case, memory replay caches are not shared across the processes, thus allowing potential for replay attacks.

KRB5_CONFIG

Allows you to change the default location of the `/etc/krb5/krb5.conf` file to enable the Kerberos library code to read configuration parameters from another file specified by KRB5_CONFIG. For example (using kinit from ksh(1)):

```
KRB5_CONFIG=/var/tmp/krb5.conf kinit
```

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWkrbu
Interface Stability	Uncommitted

See Also chmod(1), kinit(1), klist(1), ksh(1), kadmin(1M), kadmind(1M), krb5.conf(4), attributes(5), kerberos(5)

Name labels – Solaris Trusted Extensions label attributes

Description Labels are attributes that are used in mandatory policy decisions. Labels are associated, either explicitly or implicitly, with all subjects (generally processes) and objects (generally things with data such as files) that are accessible to subjects. The default Trusted Extensions mandatory policy labels are defined by a site's security administrator in `label_encodings(4)`.

Mandatory Policy

Various mandatory policies might be delivered in the lifetime of Solaris Trusted Extensions.

The default mandatory policy of Trusted Extensions is a Mandatory Access Control (MAC) policy that is equivalent to that of the Bell-LaPadula Model of the Lattice, the Simple Security Property, and the *-Property (Star Property), with restricted write up. The default mandatory policy is also equivalent to the Goguen and Mesegeur model of Non-Inteference.

For this MAC policy, two labels are always defined: `admin_low` and `admin_high`. The site's security administrator defines all other labels in `label_encodings`(4). `admin_low` is associated with all normal user readable (viewable) Trusted Extensions objects. `admin_high` is associated with all other Trusted Extensions objects. Only administrative users have MAC read (view) access to `admin_high` objects and only administrative users have MAC write (modify) access to `admin_low` objects or `admin_high` objects.

Human Readable Labels

Users interact with labels as strings. Graphical user interfaces and command line interfaces present the strings as defined in `label_encodings`(4). Human readable labels are classified at the label that they represent. Thus the string for a label `A` is only readable (viewable, translatable to or from human readable to opaque `m_label_t`) by a subject whose label allows read (view) access to that label.

Internal Text Labels

In order to store labels in publicly accessible (`admin_low`) name service databases, an unclassified internal text form is used. This textual form is not intended to be used in any interfaces other than those that are provided with the Trusted Extensions software release that created this textual form of the label.

Labels and Applications

Applications interact with labels as opaque (`m_label_t`) structures. The semantics of these opaque structures are defined by a string to `m_label_t` translation. This translation is defined in `label_encodings`(4). Various Application Programming Interfaces (API) translate between strings and `m_label_t` structures. Various APIs test access of subject-related labels to object-related labels.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	See below.

The labels implementation is Committed for systems that implement the Defense Intelligence Agency (DIA) MAC policy of `label_encodings(4)`. Other policies might exist in a future release of Trusted Extensions that might make obsolete or supplement `label_encodings`.

Internal text labels are Not-an-Interface and might change with any release of Trusted Extensions. They are intended only for input and generation on the same release of Trusted Extensions software.

As a potential porting aid for Trusted Solaris 8 applications, the opaque structure names `bslabel_t`, `blevel_t`, and `bclear_t` are defined to be equivalent to `m_label_t`. Like `m_label_t`, these types must be ported as opaque pointers. The same must be done with the various Trusted Solaris 8 label interfaces. These Trusted Solaris 8 structures and interfaces are Obsolete and might be removed from a future release of Trusted Extensions.

See Also `chk_encodings(1M)`, `blcompare(3TSOL)`, `label_to_str(3TSOL)`, `m_label_alloc(3TSOL)`, `m_label_dup(3TSOL)`, `m_label_free(3TSOL)`, `str_to_label(3TSOL)`, `label_encodings(4)`, `attributes(5)`

Bell, D. E., and LaPadula, L. J. *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MTR-2997 Rev. 2, MITRE Corp., Bedford Mass., March 1976. NTIS AD-A023 588/7.

Goguen, J. A., and Mesegeur, J.: *Security Policies and Security Models*, Proceedings 1982 Symposium on Security and Privacy, IEEE Computer Society Press, 1982, p 11-20.

Goguen, J. A., and Mesegeur, J.: *Unwinding and Interference Control*, Proceedings 1984 Symposium on Security and Privacy, IEEE Computer Society Press, 1984, p 75-86.

Compartmented Mode Workstation Labeling: Encodings Format

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

Name largefile – large file status of utilities

Description A *large file* is a regular file whose size is greater than or equal to 2 Gbyte (2^{31} bytes). A *small file* is a regular file whose size is less than 2 Gbyte.

Large file aware utilities

A utility is called *large file aware* if it can process large files in the same manner as it does small files. A utility that is large file aware is able to handle large files as input and generate as output large files that are being processed. The exception is where additional files are used as system configuration files or support files that can augment the processing. For example, the `file` utility supports the `-m` option for an alternative "magic" file and the `-f` option for a support file that can contain a list of file names. It is unspecified whether a utility that is large file aware will accept configuration or support files that are large files. If a large file aware utility does not accept configuration or support files that are large files, it will cause no data loss or corruption upon encountering such files and will return an appropriate error.

The following `/usr/bin` utilities are large file aware:

`adb`	`aliasadm`	`awk`	`bdiff`	`cat`
`chgrp`	`chmod`	`chown`	`cksum`	`cmp`
`compress`	`cp`	`csh`	`csplit`	`cut`
`dd`	`dircmp`	`du`	`egrep`	`fgrep`
`file`	`find`	`ftp`	`getconf`	`grep`
`gzip`	`head`	`join`	`jsh`	`ksh`
`ksh93`	`ln`	`ls`	`mailcompat`	`mailstats`
`mdb`	`mkdir`	`mkfifo`	`more`	`mv`
`nawk`	`page`	`paste`	`pathchck`	`pg`
`praliases`	`rcp`	`remsh`	`rksh`	`rksh93`
`rm`	`rmdir`	`rsh`	`sed`	`sh`
`sort`	`split`	`sum`	`tail`	`tar`
`tee`	`test`	`touch`	`tr`	`uncompress`
`uudcode`	`uuencode`	`vacation`	`wc`	`zcat`

The following `/usr/xpg4/bin` utilities are large file aware:

`awk`	`cp`	`chgrp`	`chown`	`du`
`egrep`	`fgrep`	`file`	`grep`	`ln`

```
ls              more            mv              rm              sed
sh              sort            tail            tr
```

The following `/usr/xpg6/bin` utilities are large file aware:

```
getconf         ls              tr
```

The following `/usr/sbin` utilities are large file aware:

```
editmap         install         makemap         mkfile          mknod
mvdir           swap
```

The following `/usr/lib` utilities are large file aware:

```
mail.local      sendmail        smrsh
```

See the USAGE section of the swap(1M) manual page for limitations of `swap` on block devices greater than 2 Gbyte on a 32–bit operating system.

The following `/usr/ucb` utilities are large file aware:

```
chown           from            ln              ls              sed
sum             touch
```

The `/usr/bin/cpio` and `/usr/bin/pax` utilities are large file aware, but cannot archive a file whose size exceeds 8 Gbyte – 1 byte.

The `/usr/bin/truss` utilities has been modified to read a dump file and display information relevant to large files, such as offsets.

cachefs file systems

The following `/usr/bin` utilities are large file aware for `cachefs` file systems:

```
cachefspack         cachefsstat
```

The following `/usr/sbin` utilities are large file aware for `cachefs` file systems:

```
cachefslog          cachefswssize       cfsadmin            fsck
mount               umount
```

nfs file systems

The following utilities are large file aware for `nfs` file systems:

```
/usr/lib/autofs/automountd              /usr/sbin/mount
/usr/lib/nfs/rquotad
```

ufs file systems

The following `/usr/bin` utility is large file aware for `ufs` file systems:

```
df
```

The following `/usr/lib/nfs` utility is large file aware for `ufs` file systems:

```
rquotad
```

The following `/usr/xpg4/bin` utility is large file aware for `ufs` file systems:

```
df
```

The following `/usr/sbin` utilities are large file aware for `ufs` file systems:

```
clri            dcopy           edquota         ff              fsck
fsdb            fsirand         fstyp           labelit         lockfs
mkfs            mount           ncheck          newfs           quot
quota           quotacheck      quotaoff        quotaon         repquota
tunefs          ufsdump         ufsrestore      umount
```

Large file safe utilities

A utility is called *large file safe* if it causes no data loss or corruption when it encounters a large file. A utility that is large file safe is unable to process properly a large file, but returns an appropriate error.

The following `/usr/bin` utilities are large file safe:

```
audioconvert        audioplay       audiorecord     comm            diff
diff3               diffmk          ed              lp              mail
mailcompat          mailstats       mailx           pack            pcat
red                 rmail           sdiff           unpack          vi
view
```

The following `/usr/xpg4/bin` utilities are large file safe:

```
ed                vi                view
```

The following `/usr/xpg6/bin` utility is large file safe:

```
ed
```

The following `/usr/sbin` utilities are large file safe:

```
lpfilter          lpforms
```

The following `/usr/ucb` utilities are large file safe:

```
Mail              lpr
```

See Also `lf64(5)`, `lfcompile(5)`, `lfcompile64(5)`

Name lf64 – transitional interfaces for 64-bit file offsets

Description The data types, interfaces, and macros described on this page provide explicit access to 64-bit file offsets. They are accessible through the transitional compilation environment described on the `lfcompile64(5)` manual page. The function prototype and semantics of a transitional interface are equivalent to those of the standard version of the call, except that relevant data types are 64-bit entities.

Data Types The following tables list the standard data or struct types in the left-hand column and their corresponding explicit 64-bit file offset types in the right-hand column, grouped by header. The absence of an entry in the left-hand column indicates that there is no existing explicit 32-bit type that corresponds to the 64–bit type listed in the right—hand column. Note that in a 64-bit application, the standard definition is equivalent to the 64-bit file offset definition.

`<aio.h>`

struct `aiocb`	struct `aiocb64`
`off_t` aio_offset;	`off64_t` aio_offset;

`<sys/dirent.h>`

struct `dirent`	struct `dirent64`
`ino_t` d_ino;	`ino64_t` d_ino;
`off_t` d_off;	`off64_t` d_off;

`<sys/fcntl.h>`

struct `flock`	struct `flock64`
`off_t` l_start;	`off64_t` l_start;
`off_t` l_len;	`off64_t` l_len;
`F_SETLK`	`F_SETLK64`
`F_SETLKW`	`F_SETLKW64`
`F_GETLK`	`F_GETLK64`
`F_FREESP`	`F_FREESP64`
`F_ALLOCSP`	`F_ALLOCSP64`
	`O_LARGEFILE`

<sys/stdio.h>

fpos_t	fpos64_t

<sys/resource.h>

rlim_t	rlim64_t
struct rlimit	struct rlimit64
rlim_t rlim_cur;	rlim64_t rlim_cur;
rlim_t rlim_max;	rlim64_t rlim_max;
RLIM_INFINITY	RLIM64_INFINITY
RLIM_SAVED_MAX	RLIM64_SAVED_MAX
RLIM_SAVED_CUR	RLIM64_SAVED_CUR

<sys/stat.h>

struct stat	struct stat64
ino_t st_ino;	ino64_t st_ino;
off_t st_size;	off64_t st_size;
blkcnt_t st_blocks;	blkcnt64_t st_blocks;

<sys/statvfs.h>

struct statvfs	struct statvfs64
fsblkcnt_t f_blocks;	fsblkcnt64_t f_blocks;
fsblkcnt_t f_bfree;	fsblkcnt64_t f_bfree;
fsblkcnt_t f_bavial;	fsblkcnt64_t f_bavial;
fsfilcnt_t f_files;	fsfilcnt64_t f_files;
fsfilcnt_t f_ffree;	fsfilcnt64_t f_ffree;
fsfilcnt_t f_favail;	fsfilcnt64_t f_favail;

<sys/types.h>

off_t;	off64_t;
ino_t;	ino64_t;
blkcnt_t;	blkcnt64_t;
fsblkcnt_t;	fsblkcnt64_t;
fsfilcnt_t;	fsfilcnt64_t;

<unistd.h>

_LFS64_LARGEFILE
_LFS64_STDIO

<sys/unistd.h>

_CS_LFS64_CFLAGS
_CS_LFS64_LDFLAGS
_CS_LFS64_LIBS
_CS_LFS64_LINTFLAGS

System Interfaces

The following tables display the standard API and the corresponding transitional interfaces for 64-bit file offsets. The interfaces are grouped by header. The interface name and the affected data types are displayed in courier font.

<aio.h>

int aio_cancel(...,	int aio_cancel64(...,
struct aiocb *);	struct aiocb64 *);
int aio_error(	int aio_error64(
const struct aiocb *);	const struct aiocb64 *);
int aio_fsync(...,	int aio_fsync64(...,
struct aiocb *);	struct aiocb64 *);
int aio_read(struct aiocb *);	int aio_read64(struct aiocb64 *);
int aio_return(struct aiocb *);	int aio_return64(struct aiocb64 *);
int aio_suspend(	int aio_suspend64(

const struct aiocb *, ...);	const struct aiocb64 *, ...);
int aio_waitn(aiocb_t *[],	int aio_waitn64(aiocb64_t *[],
...);	...);
int aio_write(struct aiocb *);	int aio_write64(struct aiocb64 *);
int lio_listio(...,	int lio_listio64(...,
const struct aiocb *, ...);	const struct aiocb64 *, ...);

<dirent.h>

int alphasort(	int alphasort64(
const struct dirent **,	const struct dirent64 **,
const struct dirent **)	const struct dirent64 **)
struct dirent *readdir();	struct dirent64 *readdir64();
struct dirent *readdir_r();	struct dirent64 *readdir64_r();
int scandir(...,	int scandir64(...,
struct dirent *(*[]),	struct dirent64 *(*[]),
int (*)(const struct dirent *),	int (*)(const struct dirent64 *),
int (*)(const struct dirent **,	int (*)(const struct dirent64 **,
const struct dirent **))	const struct dirent64 **))

<fcntl.h>

int attropen();	int attropen64();
int creat();	int creat64();
int open();	int open64();
int openat();	int openat64();
int posix_fadvise()	int posix_fadvise64()
int posix_fallocate()	int posix_fallocate64()

<ftw.h>

int ftw(...,	int ftw64(...,
const struct stat *, ...);	const struct stat64 *, ...);
int nftw(..	int nftw64(...,
const struct stat *, ...);	const struct stat64 *, ...);

<libgen.h>

char *copylist(..., off_t);	char *copylist64(..., off64_t);

<stdio.h>

int fgetpos();	int fgetpos64();
FILE *fopen();	FILE *fopen64();
FILE *freopen();	FILE *freopen64();
int fseeko(..., off_t, ...);	int fseeko64(..., off64_t, ...);
int fsetpos(...,	int fsetpos64(...,
const fpos_t *);	const fpos64_t *);
off_t ftello();	off64_t ftello64()();
FILE *tmpfile();	FILE *tmpfile64();

<stdlib.h>

int mkstemp();	int mkstemp64();

<sys/async.h>

int aioread(..., off_t, ...);	int aioread64(..., off64_t, ...);
int aiowrite(..., off_t, ...);	int aiowrite64(..., off64_t, ...);

<sys/dirent.h>

int getdents(..., dirent);	int getdents64(..., dirent64);

<sys/mman.h>

void mmap(..., off_t);	void mmap64(..., off64_t);

<sys/resource.h>

int getrlimit(...,	int getrlimit64(...,
struct rlimit *);	struct rlimit64 *);
int setrlimit(...,	int setrlimit64(...,
const struct rlimit *);	const struct rlimit64 *);

<sys/sendfile.h>

ssize_t sendfile(...,	ssize_t sendfile64(...,
off_t *, ...);	off64_t *, ...);
ssize_t sendfilev(..., const	ssize_t sendfilev64(..., const
struct sendfilevec *, ...);	struct sendfilevec64 *, ...);

<sys/stat.h>

int fstat(..., struct stat *);	int fstat64(..., struct stat64 *);
int fstatat(...,	int fstatat64(...,
struct stat *, int);	struct stat64 *, int);
int lstat(..., struct stat *);	int lstat64(..., struct stat64 *);
int stat(..., struct stat *);	int stat64(..., struct stat64 *);

<sys/statvfs.h>

int statvfs(...,	int statvfs64(...,
struct statvfs *);	struct statvfs64 *);
int fstatvfs(...,	int fstatvfs64(...,
struct statvfs *);	struct statvfs64 *);

<ucbinclude/stdio.h>

FILE *fopen()	FILE *fopen64()
FILE *freopen()	FILE *freopen64()

<ucbinclude/sys/dir.h>

int alphasort(	int alphasort64(
struct direct **,	struct direct64 **,
struct direct **);	struct direct64 **);
struct direct *readdir();	struct direct64 *readdir64();
int scandir(...,	int scandir64(...,
struct direct *(*[]);, ...);	struct direct64 *(*[]);, ...);

<unistd.h>

int lockf(..., off_t);	int lockf64(..., off64_t);
off_t lseek(..., off_t, ...);	off64_t lseek64(..., off64_t, ...);
int ftruncate(..., off_t);	int ftruncate64..., off64_t);
ssize_t pread(..., off_t);	ssize_t pread64..., off64_t);
ssize_t pwrite(..., off_t);	ssize_t pwrite64(..., off64_t);
int truncate(..., off_t);	int truncate64(..., off64_t);

See Also lfcompile(5), lfcompile64(5)

Name lfcompile – large file compilation environment for 32-bit applications

Description All 64-bit applications can manipulate large files by default. The methods described on this page allow 32-bit applications to manipulate large files.

In the large file compilation environment, source interfaces are bound to appropriate 64-bit functions, structures, and types. Compiling in this environment allows 32-bit applications to access files whose size is greater than or equal to 2 Gbyte (2^{31} bytes).

Each interface named *xxx*`()` that needs to access 64-bit entities to access large files maps to a *xxx*`64()` call in the resulting binary. All relevant data types are defined to be of correct size (for example, `off_t` has a typedef definition for a 64-bit entity).

An application compiled in this environment is able to use the *xxx*`()` source interfaces to access both large and small files, rather than having to explicitly utilize the transitional *xxx*`64()` interface calls to access large files. See the `lfcompile64(5)` manual page for information regarding the transitional compilation environment.

Applications can be compiled in the large file compilation environment by using the following methods:

- Use the `getconf(1)` utility with one or more of the arguments listed in the table below. This method is recommended for portable applications.

argument	purpose
`LFS_CFLAGS`	obtain compilation flags necessary to enable the large file compilation environment
`LFS_LDFLAGS`	obtain link editor options
`LFS_LIBS`	obtain link library names
`LFS_LINTFLAGS`	obtain lint options

- Set the compile-time flag `_FILE_OFFSET_BITS` to 64 before including any headers. Applications may combine objects produced in the large file compilation environment with objects produced in the transitional compilation environment, but must be careful with respect to interoperability between those objects. Applications should not declare global variables of types whose sizes change between compilation environments.

Access to Additional Large File Interfaces The `fseek()` and `ftell()` functions *do not* map to functions named `fseek64()` and `ftell64()`; rather, the large file additions `fseeko()` and `ftello()`, have functionality identical to `fseek()` and `ftell()` and *do* map to the 64-bit functions `fseeko64()` and `ftello64()`. Applications wishing to access large files should use `fseeko()` and `ftello()` in place of `fseek()` and `ftell()`. See the `fseek(3C)` and `ftell(3C)` manual pages for information about `fseeko()` and `ftello()`.

Applications wishing to access `fseeko()` and `ftello()` as well as the POSIX and X/Open specification-conforming interfaces should define the macro `_LARGEFILE_SOURCE` to be 1 and set whichever feature test macros are appropriate to obtain the desired environment (see `standards(5)`).

Examples

In the following examples, the large file compilation environment is accessed by invoking the `getconf` utility with one of the arguments listed in the table above. The additional large file interfaces are accessed by specifying `–D_LARGEFILE_SOURCE`.

The examples that use the form of command substitution specifying the command within parentheses preceded by a dollar sign can be executed only in a POSIX-conforming shell such as the Korn Shell (see `ksh(1)`). In a shell that is not POSIX-conforming, such as the Bourne Shell (see `sh(1)`) and the C Shell (see `csh(1)`), the `getconf` calls must be enclosed within grave accent marks, as shown in the second example.

EXAMPLE 1 Compile a program with a "large" `off_t` that uses `fseeko()`, `ftello()`, and `yacc`.

The following example compiles a program with a "large" `off_t` and uses `fseeko()`, `ftello()`, and `yacc(1)`.

```
$ c89 -D_LARGEFILE_SOURCE                \
      -D_FILE_OFFSET_BITS=64 -o foo      \
      $(getconf LFS_CFLAGS) y.tab.c b.o  \
      $(getconf LFS_LDFLAGS)             \
      -ly $(getconf LFS_LIBS)
```

EXAMPLE 2 Compile a program with a "large" `off_t` that does not use `fseeko()` and `ftello()` and has no application specific libraries.

```
% c89 -D_FILE_OFFSET_BITS=64     \
     ` getconf LFS_CFLAG`S a.c   \
     ` getconf LFS_LDFLAG`S      \
     ` getconf LFS_LIB`S         \
```

EXAMPLE 3 Compile a program with a "default" `off_t` that uses `fseeko()` and `ftello()`.

```
$ c89 –D_LARGEFILE_SOURCE  a.c
```

See Also

`csh(1)`, `getconf(1)`, `ksh(1)`, `yacc(1)`, `sh(1)`, `fseek(3C)`, `ftell(3C)`, `lf64(5)`, `lfcompile64(5)`, `standards(5)`

Notes

Certain system-specific or non-portable interfaces are not usable in the large file compilation environment. Known cases are:

- Kernel data structures read from `/dev/kmem`.
- Interfaces in the kernel virtual memory library, `–lkvm`.
- Interfaces in the ELF access library, `–lelf`.
- Interfaces to `/proc` defined in `<procfs.h>`.
- The `ustat(2)` system call.

Programs that use these interfaces should not be compiled in the large file compilation environment. As a partial safeguard against making this mistake, including either of the <libelf.h> or <sys/procfs.h> header files will induce a compilation error when the large file compilation environment is enabled.

In general, caution should be exercised when using any separately-compiled library whose interfaces include data items of type `off_t` or the other redefined types either directly or indirectly, such as with '`struct stat`'. (The redefined types are `off_t`, `rlim_t`, `ino_t`, `blkcnt_t`, `fsblkcnt_t`, and `fsfilcnt_t`.) For the large file compilation environment to work correctly with such a library, the library interfaces must include the appropriate *xxx*`64()` binary entry points and must have them mapped to the corresponding primary functions when `_FILE_OFFSET_BITS` is set to 64.

Care should be exercised using any of the `printf()` or `scanf()` routines on variables of the types mentioned above. In the large file compilation environment, these variables should be printed or scanned using `long long` formats.

Bugs Symbolic formats analogous to those found in <sys/int_fmtio.h> do not exist for printing or scanning variables of the types that are redefined in the large file compilation environment.

Name lfcompile64 – transitional compilation environment

Description All 64-bit applications can manipulate large files by default. The transitional interfaces described on this page can be used by 32-bit and 64-bit applications to manipulate large files.

In the transitional compilation environment, explicit 64-bit functions, structures, and types are added to the API. Compiling in this environment allows both 32-bit and 64-bit applications to access files whose size is greater than or equal to 2 Gbyte (2^{31} bytes).

The transitional compilation environment exports all the explicit 64-bit functions (*xxx*`64()`) and types in addition to all the regular functions (*xxx*`()`) and types. Both *xxx*`()` and *xxx*`64()` functions are available to the program source. A 32-bit application must use the *xxx*`64()` functions in order to access large files. See the `lf64(5)` manual page for a complete listing of the 64-bit transitional interfaces.

The transitional compilation environment differs from the large file compilation environment, wherein the underlying interfaces are bound to 64-bit functions, structures, and types. An application compiled in the large file compilation environment is able to use the *xxx*`()` source interfaces to access both large and small files, rather than having to explicitly utilize the transitional *xxx*`64()` interface calls to access large files. See the `lfcompile(5)` manual page for more information regarding the large file compilation environment.

Applications may combine objects produced in the large file compilation environment with objects produced in the transitional compilation environment, but must be careful with respect to interoperability between those objects. Applications should not declare global variables of types whose sizes change between compilation environments.

For applications that do not wish to conform to the POSIX or X/Open specifications, the 64-bit transitional interfaces are available by default. No compile-time flags need to be set.

Access to Additional Large File Interfaces

Applications that wish to access the transitional interfaces as well as the POSIX or X/Open specification-conforming interfaces should use the following compilation methods and set whichever feature test macros are appropriate to obtain the desired environment (see `standards(5)`).

- Set the compile-time flag `_LARGEFILE64_SOURCE` to 1 before including any headers.
- Use the `getconf(1)` command with one or more of the following arguments:

argument	purpose
`LFS64_CFLAGS`	obtain compilation flags necessary to enable the transitional compilation environment
`LFS64_LDFLAGS`	obtain link editor options
`LFS64_LIBS`	obtain link library names
`LFS64_LINTFLAGS`	obtain lint options

Examples

In the following examples, the transitional compilation environment is accessed by invoking the `getconf` utility with one of the arguments listed in the table above. The additional large file interfaces are accessed either by specifying `–D_LARGEFILE64_SOURCE` or by invoking the `getconf` utility with the arguments listed above.

The example that uses the form of command substitution specifying the command within parentheses preceded by a dollar sign can be executed only in a POSIX-conforming shell such as the Korn Shell (see `ksh(1)`). In a shell that is not POSIX-conforming, such as the Bourne Shell (see `sh(1)`) and the C Shell (see `csh(1)`), the command must be enclosed within grave accent marks.

EXAMPLE 1 An example of compiling a program using transitional interfaces such as `lseek64()` and `fopen64()`:

```
$ c89 -D_LARGEFILE64_SOURCE        \
      $(getconf LFS64_CFLAGS) a.c  \
      $(getconf LFS64_LDFLAGS)     \
      $(getconf LFS64_LIBS)
```

EXAMPLE 2 An example of running lint on a program using transitional interfaces:

```
% lint -D_LARGEFILE64_SOURCE        \
     ` getconf LFS64_LINTFLAG`S |PO \
     ` getconf LFS64_LIB`S
```

See Also

`getconf(1)`, `lseek(2)`, `fopen(3C)`, `lf64(5)`, `standards(5)`

Name live_upgrade – overview of Live Upgrade feature

Description The Live Upgrade feature of the Solaris operating environment enables you to maintain multiple operating system images on a single system. An image—called a boot environment, or BE—represents a set of operating system and application software packages. The BEs might contain different operating system and/or application versions.

On a system with the Solaris Live Upgrade software, your currently booted OS environment is referred to as your active, or current BE. You have one active, or current BE; all others are inactive. You can perform any number of modifications to inactive BEs on the same system, then boot from one of those BEs. If there is a failure or some undesired behavior in the newly booted BE, Live Upgrade software makes it easy for you to fall back to the previously running BE.

Live Upgrade software includes a full suite of commands, listed below and described in individual man pages, which implement all of the Live Upgrade features and functions.

The following are some of the tasks you can perform with Live Upgrade software:

- You can make one or more copies of the currently running system.
- You can upgrade to a new OS version on a second boot environment, then boot from that environment. If you choose, you can then fall back to your original boot environment or boot from yet another environment.
- You can install application or OS packages to a boot environment, then boot from that environment.
- You can install OS patches to a boot environment, then boot from that environment.
- From a flash archive, you can install an OS to a boot environment, then boot from that environment. See `flar(1M)` for information on administering flash archives.
- You can split and rejoin file systems in a new BE. For example, you can separate `/usr`, `/var`, and `/opt` from `/`, putting them on their own partitions. Conversely, you could join these file systems on a single partition under `/`.
- You can mount any or all of the filesystems of a BE that is not active, compare the files in any pair of BEs, delete or rename a BE, and perform other administrative tasks.

The Live Upgrade software supports upgrade from any valid Solaris installation medium, including a CD-ROM, an NFS or UFS directory, or a flash archive. (See `flash_archive(4)` for a description of the flash archive feature.)

In simplest terms, a BE, for Live Upgrade, consists of the disk slice containing a root file system and the file system/device (usually disk) slice entries specified in `vfstab(4)`. This set of slices is not limited to a single disk. This means that you can have multiple BEs on a single device, or have a BE spread across slices on multiple devices. The BE includes any non-global `zones(5)` that might exist on the system as well. If any of the non-global zones in the BE have separate file systems, the disk slices making up these file systems are considered part of the BE.

The minimal requirement for a Live Upgrade BE is the same as for any Solaris boot environment: you must have root (`/`) and `usr` filesystems (which might both reside on `/`). All filesystems except for `/`, `/usr`, `/var`, and `/opt` can be shared among multiple BEs, if you choose.

Each BE must have a unique copy of the file systems that contain the OS—`/`, `/usr`, `/var`, and `/opt`. For Live Upgrade purposes, these are referred to as non–shareable (sometimes referred to as *critical*) file systems. With other file systems, such as `/export` or `/home`, you have the option of copying the files to a new BE or, the default, sharing them among BEs. These are referred to as shareable file systems. A BE is made up of a unique copy of one or more non–shareable file systems and zero or more copies of shareable file systems.

Live Upgrade commands support an option (`-X`) that enables XML output. Characteristics of the XML are specified in a DTD shipped with the product. XML output enables programmatic parsing of portions of the command output.

Live Upgrade supports the notion of a BE description, an optional attribute of a BE. A BE description can be of any length and format. It might be a text string or a binary file. See `ludesc(1M)` for details.

Below is an example set of steps that you might follow in the use of Live Upgrade software. This example is by no means exhaustive of the possibilities of the use of the Live Upgrade software.

1. You create a new BE, using `lucreate(1M)`. The first time you create a BE on a given system, you must designate the current Solaris operating environment as a BE (give it a name). You then specify a name and a set of device (disk) slices you want to use for the new BE. The `lucreate` command copies the contents of the current Solaris operating environment (now a BE) to the new BE.

   After you have created additional BEs, you can use a BE other than the current BE as the source for a new BE. Also, you can create an empty BE onto which you can later install a flash archive.
2. Using `luupgrade(1M)`, you upgrade the OS version on your new BE (or on yet another BE you created with `lucreate`). The `luupgrade` enables you to upgrade an OS (from any valid Solaris installation medium, including a flash archive), add or remove packages (OS or application), and add or remove patches.
3. You use `luactivate(1M)` to make the new BE bootable. The next time you reboot your system, you will come up in the new BE.
4. Using `lucompare(1M)`, you compare the system files on two different BEs. This utility gives you a comprehensive list of the files that have differences.
5. Using `lumount(1M)`, you mount the filesystems of a BE that is not active, enabling you to make changes. When you are finished with the changes, use `lumount(1M)` to unmount the BE's file systems.

6. Upon booting a new BE, you discover a failure or some other undesirable behavior. Using the procedure specified in `luactivate`, you can fall back to the previous BE.
7. Using `ludelete` then `lucreate`, you reassign file systems on the now-deleted BE to different disk slices. You separate `/opt` and `/var` from `/` on the new BE. Also, you specify that swap be spread over slices on multiple disks.

The following is a summary of Live Upgrade commands. All commands require root privileges.

`lu`	FMLI-based interface for creating and administering BEs. No longer recommended for customer use.
`luactivate`	Designate a BE as the BE to boot from upon the next reboot of the system.
`lucancel`	Cancel a previously scheduled operation.
`lucompare`	Compare the contents of two BEs.
`lucreate`	Create a BE.
`lucurr`	Display the name of the current BE.
`ludelete`	Delete a BE.
`ludesc`	Add or change BE descriptions.
`lufslist`	List the file systems on a specified BE.
`lumake`	Re-create a BE based on the active BE.
`lumount`, `luumount`	Mount, unmount file systems of a specified BE.
`lurename`	Rename a BE.
`lustatus`	For all BEs on a system, report on whether a BE is active, active upon the next reboot, in the midst of a copy operation, and whether a copy operation is scheduled for it.
`luupgrade`	Upgrade an OS and install application software on a BE. Such software includes flash archives, complete OS installations, OS and application packages, and OS patches.

Files `/etc/lutab` list of BEs on the system

See Also `luactivate(1M)`, `lucancel(1M)`, `lucompare(1M)`, `lucreate(1M)`, `lucurr(1M)`, `ludelete(1M)`, `ludesc(1M)`, `lufslist(1M)`, `lumake(1M)`, `lumount(1M)`, `lurename(1M)`, `lustatus(1M)`, `luupgrade(1M)`, `lutab(4)`, `zones(5)`

Notes Correct operation of Solaris Live Upgrade requires that a limited set of patch revisions be installed for a given OS version. Before installing or running Live Upgrade, you are required to

install the limited set of patch revisions. Make sure you have the most recently updated patch list by consulting `http://sunsolve.sun.com`. Search for the infodoc 72099 on the SunSolve web site.

It is possible for an operating system upgrade to remove installed patches. Prior to such an upgrade, use `analyze_patches`, as described in `luupgrade(1M)`, to determine which, if any, patches will be removed.

For versions of the Solaris operating system prior to Solaris 10, Live Upgrade supports the release it is distributed on and up to three marketing releases back. For example, if you obtained Live Upgrade with Solaris 9 (including a Solaris 9 upgrade), that version of Live Upgrade supports Solaris versions 2.6, Solaris 7, and Solaris 8, in addition to Solaris 9. No version of Live Upgrade supports a Solaris version prior to Solaris 2.6.

Starting with version 10 of the Solaris operating system, Live Upgrade supports the release it is distributed on and up to two marketing releases back. For example, if you obtained Live Upgrade with Solaris 10 (including a Solaris 10 upgrade), that version of Live Upgrade supports Solaris 8 and Solaris 9, in addition to Solaris 10.

Name locale – subset of a user's environment that depends on language and cultural conventions

Description A `locale` is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

`LC_CTYPE`	Character classification and case conversion.
`LC_COLLATE`	Collation order.
`LC_TIME`	Date and time formats.
`LC_NUMERIC`	Numeric formatting.
`LC_MONETARY`	Monetary formatting.
`LC_MESSAGES`	Formats of informative and diagnostic messages and interactive responses.

The standard utilities base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions will also be modified based on the current locale, as defined by the last call to `setlocale(3C)`.

Locales other than those supplied by the implementation can be created by the application via the `localedef(1)` utility. The value that is used to specify a locale when using environment variables will be the string specified as the *name* operand to `localedef` when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale.

Applications can select the desired locale by invoking the `setlocale()` function with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the `setlocale()` function sets the appropriate environment.

Locale Definition Locales can be described with the file format accepted by the `localedef` utility.

The locale definition file must contain one or more locale category source definitions, and must not contain more than one definition for the same locale category.

A category source definition consists of a category header, a category body and a category trailer. A category header consists of the character string naming of the category, beginning with the characters `LC_`. The category trailer consists of the string `END`, followed by one or more blank characters and the string used in the corresponding category header.

The category body consists of one or more lines of text. Each line contains an identifier, optionally followed by one or more operands. Identifiers are either keywords, identifying a particular locale element, or collating elements. Each keyword within a locale must have a unique name (that is, two categories cannot have a commonly-named keyword). No keyword can start with the characters `LC_`. Identifiers must be separated from the operands by one or more blank characters.

Operands must be characters, collating elements, or strings of characters. Strings must be enclosed in double-quotes (`"`). Literal double-quotes within strings must be preceded by the <*escape character*>, as described below. When a keyword is followed by more than one operand, the operands must be separated by semicolons (`;`). Blank characters are allowed both before and after a semicolon.

The first category header in the file can be preceded by a line modifying the comment character. It has the following format, starting in column 1:

`"comment_char %c\n",`<*comment character*>

The comment character defaults to the number sign (#). Blank lines and lines containing the <*comment character*> in the first position are ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It has the following format, starting in column 1:

`"escape_char %c\n",`<*escape character*>

The escape character defaults to backslash.

A line can be continued by placing an escape character as the last character on the line; this continuation character will be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding `{LINE_MAX}` bytes, it places no limits on the accumulated length of the continued line. Comment lines cannot be continued on a subsequent line using an escaped newline character.

Individual characters, characters in strings, and collating elements must be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal or decimal constants. When non-symbolic notation is used, the resultant locale definitions will in many cases not be portable between systems. The left angle bracket (<) is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it must be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets < and >. The symbolic name, including the angle brackets, must exactly match a symbolic name defined in the charmap file specified via the `localedef -f` option, and will be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file constitutes an

error, unless the category is LC_CTYPE or LC_COLLATE, in which case it constitutes a warning condition (see localedef(1) for a description of action resulting from errors and warnings). The specification of a symbolic name in a collating-element or collating-symbol section that duplicates a symbolic name in the charmap file (if present) is an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

Example:

```
<C>;<c-cedilla> "<M><a><y>"
```

2. A character can be represented by the character itself, in which case the value of the character is implementation-dependent. Within a string, the double-quote character, the escape character and the right angle bracket character must be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters

   , ; < > *escape_char*

   must be escaped to be interpreted as the character itself.

   Example:

   ```
   c "May"
   ```

3. A character can be represented as an octal constant. An octal constant is specified as the escape character followed by two or more octal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   Example:

   ```
   \143;\347;\143\150 "\115\141\171"
   ```

4. A character can be represented as a hexadecimal constant. A hexadecimal constant is specified as the escape character followed by an x followed by two or more hexadecimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   Example:

   ```
   \x63;\xe7;\x63\x68 "\x4d\x61\x79"
   ```

5. A character can be represented as a decimal constant. A decimal constant is specified as the escape character followed by a d followed by two or more decimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   Example:

   ```
   \d99;\d231;\d99\d104 "\d77\d97\d121"
   ```

Only characters existing in the character set for which the locale definition is created can be specified, whether using symbolic names, the characters themselves, or octal, decimal or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal or hexadecimal constants. Symbolic names not present in the charmap file can be specified and will be ignored, as specified under item 1 above.

LC_CTYPE The LC_CTYPE category defines character classification, case conversion and other character attributes. In addition, a series of characters can be represented by three adjacent periods representing an ellipsis symbol (. . .). The ellipsis specification is interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification is valid only within a single encoded character set, that is, within a group of characters of the same size. An ellipsis is interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

Example:

```
\x30;...;\x39;
```

includes in the character class all characters with encoded values between the endpoints.

The following keywords are recognized. In the descriptions, the term ‘‘automatically included" means that it is not an error either to include or omit any of the referenced characters.

The character classes `digit`, `xdigit`, `lower`, `upper`, and `space` have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differ from the implementation default values.

`upper` Define characters to be classified as upper-case letters.

In the POSIX locale, the 26 upper-case letters are included:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

In a locale definition file, no character specified for the keywords `cntrl`, `digit`, `punct`, or `space` can be specified. The upper-case letters `A` to `Z` are automatically included in this class.

`lower` Define characters to be classified as lower-case letters. In the POSIX locale, the 26 lower-case letters are included:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

In a locale definition file, no character specified for the keywords `cntrl`, `digit`, `punct`, or `space` can be specified. The lower-case letters `a` to `z` of the portable character set are automatically included in this class.

`alpha` Define characters to be classified as letters.

In the POSIX locale, all characters in the classes `upper` and `lower` are included.

In a locale definition file, no character specified for the keywords `cntrl`, `digit`, `punct`, or `space` can be specified. Characters classified as either `upper` or `lower` are automatically included in this class.

`digit` Define the characters to be classified as numeric digits.

In the POSIX locale, only

```
0 1 2 3 4 5 6 7 8 9
```

are included.

In a locale definition file, only the digits `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, and `9` can be specified, and in contiguous ascending sequence by numerical value. The digits `0` to `9` of the portable character set are automatically included in this class.

The definition of character class `digit` requires that only ten characters; the ones defining digits can be specified; alternative digits (for example, Hindi or Kanji) cannot be specified here.

`alnum` Define characters to be classified as letters and numeric digits. Only the characters specified for the `alpha` and `digit` keywords are specified. Characters specified for the keywords `alpha` and `digit` are automatically included in this class.

`space` Define characters to be classified as white-space characters.

In the POSIX locale, at a minimum, the characters `SPACE`, `FORMFEED`, `NEWLINE`, `CARRIAGE RETURN`, `TAB`, and `VERTICAL TAB` are included.

In a locale definition file, no character specified for the keywords `upper`, `lower`, `alpha`, `digit`, `graph`, or `xdigit` can be specified. The characters `SPACE`, `FORMFEED`, `NEWLINE`, `CARRIAGE RETURN`, `TAB`, and `VERTICAL TAB` of the portable character set, and any characters included in the class `blank` are automatically included in this class.

`cntrl` Define characters to be classified as control characters.

In the POSIX locale, no characters in classes `alpha` or `print` are included.

In a locale definition file, no character specified for the keywords `upper`, `lower`, `alpha`, `digit`, `punct`, `graph`, `print`, or `xdigit` can be specified.

`punct` Define characters to be classified as punctuation characters.

In the POSIX locale, neither the space character nor any characters in classes `alpha`, `digit`, or `cntrl` are included.

In a locale definition file, no character specified for the keywords `upper`, `lower`, `alpha`, `digit`, `cntrl`, `xdigit` or as the space character can be specified.

`graph` Define characters to be classified as printable characters, not including the space character.

In the POSIX locale, all characters in classes `alpha`, `digit`, and `punct` are included; no characters in class `cntrl` are included.

In a locale definition file, characters specified for the keywords `upper`, `lower`, `alpha`, `digit`, `xdigit`, and `punct` are automatically included in this class. No character specified for the keyword `cntrl` can be specified.

`print` Define characters to be classified as printable characters, including the space character.

In the POSIX locale, all characters in class `graph` are included; no characters in class `cntrl` are included.

In a locale definition file, characters specified for the keywords `upper`, `lower`, `alpha`, `digit`, `xdigit`, `punct`, and the space character are automatically included in this class. No character specified for the keyword `cntrl` can be specified.

`xdigit` Define the characters to be classified as hexadecimal digits.

In the POSIX locale, only:

`0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f`

are included.

In a locale definition file, only the characters defined for the class `digit` can be specified, in contiguous ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 to 15 inclusive, with each set in ascending order (for example `A`, `B`, `C`, `D`, `E`, `F`, `a`, `b`, `c`, `d`, `e`, `f`). The digits `0` to `9`, the upper-case letters `A` to `F` and the lower-case letters `a` to `f` of the portable character set are automatically included in this class.

The definition of character class `xdigit` requires that the characters included in character class `digit` be included here also.

`blank` Define characters to be classified as blank characters.

In the POSIX locale, only the space and tab characters are included.

In a locale definition file, the characters space and tab are automatically included in this class.

`charclass` Define one or more locale-specific character class names as strings separated by semi-colons. Each named character class can then be defined subsequently in the `LC_CTYPE` definition. A character class name consists of at least one and at most `{CHARCLASS_NAME_MAX}` bytes of alphanumeric characters from the portable filename character set. The first character of a character class name cannot be a digit. The name cannot match any of the `LC_CTYPE` keywords defined in this document.

`charclass-name` Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, the locale-specific named character classes need not exist. If a class name is defined by a `charclass` keyword, but no characters are subsequently assigned to it, this is not an error; it represents a class without any characters belonging to it. The `charclass-name` can be used as the *property* argument to the `wctype(3C)` function, in regular expression and shell pattern-matching bracket expressions, and by the `tr(1)` command.

`toupper` Define the mapping of lower-case letters to upper-case letters.

In the POSIX locale, at a minimum, the 26 lower-case characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

are mapped to the corresponding 26 upper-case characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

In a locale definition file, the operand consists of character pairs, separated by semicolons. The characters in each character pair are separated by a comma and the pair enclosed by parentheses. The first character in each pair is the lower-case letter, the second the corresponding upper-case letter. Only characters specified for the keywords `lower` and `upper` can be specified. The lower-case letters `a` to `z`, and their corresponding upper-case letters `A` to `Z`, of the portable character set are automatically included in this mapping, but only when the `toupper` keyword is omitted from the locale definition.

`tolower` Define the mapping of upper-case letters to lower-case letters.

In the POSIX locale, at a minimum, the 26 upper-case characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

are mapped to the corresponding 26 lower-case characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

In a locale definition file, the operand consists of character pairs, separated by semicolons. The characters in each character pair are separated by a comma and the pair enclosed by parentheses. The first character in each pair is the upper-case letter, the second the corresponding lower-case letter. Only characters specified for the keywords `lower` and `upper` can be specified. If the `tolower` keyword is omitted from the locale definition, the mapping will be the reverse mapping of the one specified for `toupper`.

LC_COLLATE The `LC_COLLATE` category provides a collation sequence definition for numerous utilities (such as `sort(1)`, `uniq(1)`, and so forth), regular expression matching (see `regex(5)`), and the `strcoll(3C)`, `strxfrm(3C)`, `wcscoll(3C)`, and `wcsxfrm(3C)` functions.

A collation sequence definition defines the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values, that is, by assigning each element one or more collation values (also known as collation weights). The following capabilities are provided:

1. `Multi-character collating elements`. Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).
2. `User-defined ordering of collating elements`. Each collating element is assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicity specified, also as the collation weight to be used in sorting.
3. `Multiple weights and equivalence classes`. Collating elements can be assigned one or more (up to the limit `{COLL_WEIGHTS_MAX}`) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.
4. `One-to-Many mapping`. A single character is mapped into a string of collating elements.
5. `Equivalence class definition`. Two or more collating elements have the same collation value (primary weight).
6. `Ordering by weights`. When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements. The elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, the pairs of collating elements are recompared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords are recognized in a collation sequence definition. They are described in detail in the following sections.

`copy`	Specify the name of an existing locale which is used as the definition of this category. If this keyword is specified, no other keyword is specified.
`collating-element`	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
`collating-symbol`	Define a collating symbol for use in collation order statements. This keyword is optional.
`order_start`	Define collation rules. This statement is followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
`order_end`	Specify the end of the collation-order statements.

collating-element *keyword*

In addition to the collating elements in the character set, the `collating-element` keyword is used to define multi-character collating elements. The syntax is:

`"collating-element %s from \"%s\"\n",<`*collating-symbol*`>,<`*string*`>`

The <*collating-symbol*> operand is a symbolic name, enclosed between angle brackets (< and >), and must not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <*collating-element*> defined via this keyword is only recognized with the `LC_COLLATE` category.

Example:

`collating-element` <ch> from "<c><h>"
`collating-element` <e-acute> from "<acute><e>"
`collating-element` <ll> from "ll"

collating-symbol *keyword*

This keyword will be used to define symbols for use in collation sequence statements; that is, between the `order_start` and the `order_end` keywords. The syntax is:

`"collating-symbol %s\n",<`*collating-symbol*`>`

The <*collating-symbol*> is a symbolic name, enclosed between angle brackets (< and >), and must not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition.

A `collating-symbol` defined via this keyword is only recognized with the `LC_COLLATE` category.

Example:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

The `collating-symbol` keyword defines a symbolic name that can be associated with a relative position in the character order sequence. While such a symbolic name does not represent any collating element, it can be used as a weight.

order_start *keyword* The `order_start` keyword must precede collation order entries and also defines the number of weights for this collation sequence definition and other collation rules.

The syntax of the `order_start` keyword is:

`"order_start %s;%s;...;%s\n",<`*sort-rules*`>,<`*sort-rules*`>`

The operands to the `order_start` keyword are optional. If present, the operands define rules to be applied when strings are compared. The number of operands define how many weights each element is assigned. If no operands are present, one `forward` operand is assumed. If present, the first operand defines rules to be applied when comparing strings using the first (primary) weight; the second when comparing strings using the second weight, and so on. Operands are separated by semicolons (`;`). Each operand consists of one or more collation directives, separated by commas (`,`). If the number of operands exceeds the `{COLL_WEIGHTS_MAX}` limit, the utility will issue a warning message. The following directives will be supported:

`forward` Specifies that comparison operations for the weight level proceed from start of string towards the end of string.

`backward` Specifies that comparison operations for the weight level proceed from end of string towards the beginning of string.

`position` Specifies that comparison operations for the weight level will consider the relative position of elements in the strings not subject to `IGNORE`. The string containing an element not subject to `IGNORE` after the fewest collating elements subject to `IGNORE` from the start of the compare will collate first. If both strings contain a character not subject to `IGNORE` in the same relative position, the collating values assigned to the elements will determine the ordering. In case of equality, subsequent characters not subject to `IGNORE` are considered in the same manner.

The directives `forward` and `backward` are mutually exclusive.

Example:

```
order_start    forward;backward
```

If no operands are specified, a single `forward` operand is assumed.

Collation Order

The `order_start` keyword is followed by collating identifier entries. The syntax for the collating element entries is:

"`%s %s;%s;...;%s\n`"<*collating-identifier*>,<*weight*>,<*weight*>,...

Each *collating-identifier* consists of either a character described in `Locale Definition` above, a <*collating-element*>, a <*collating-symbol*>, an ellipsis, or the special symbol `UNDEFINED`. The order in which collating elements are specified determines the character order sequence, such that each collating element compares less than the elements following it. The NUL character compares lower than any other character.

A <*collating-element*> is used to specify multi-character collating elements, and indicates that the character sequence specified via the <*collating-element*> is to be collated as a unit and in the relative order specified by its place.

A <*collating-symbol*> is used to define a position in the relative order for use in weights. No weights are specified with a <*collating-symbol*>.

The ellipsis symbol specifies that a sequence of characters will collate according to their encoded character values. It is interpreted as indicating that all characters with a coded character set value higher than the value of the character in the preceding line, and lower than the coded character set value for the character in the following line, in the current coded character set, will be placed in the character collation order between the previous and the following character in ascending order according to their coded character set values. An initial ellipsis is interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the following line specified the highest coded character set value in the current coded character set. An ellipsis is treated as invalid if the preceding or following lines do not specify characters in the current coded character set. The use of the ellipsis symbol ties the definition to a specific coded character set and may preclude the definition from being portable beween implementations.

The symbol `UNDEFINED` is interpreted as including all coded character set values not specified explicitly or via the ellipsis symbol. Such characters are inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no `UNDEFINED` symbol is specified, and the current coded character set contains characters not specified in this section, the utility will issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element are used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same *equivalence class* if they have the same primary weight. Collation behaves as if, for each weight level, elements subject to `IGNORE` are removed, unless the `position` collation directive is specified for the corresponding level with the `order_start` keyword. Then each successive pair of elements is

compared according to the relative weights for the elements. If the two strings compare equal, the process is repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights are expressed as characters described in Locale Definition above, <*collating-symbol*>s, <*collating-element*>s, an ellipsis, or the special symbol IGNORE. A single character, a <*collating–symbol*> or a <*collating–element*> represent the relative position in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the character <eszet> is given the string "<s><s>" as a weight, comparisons are performed as if all occurrences of the character <eszet> are replaced by <s><s> (assuming that <s> has the collating weight <s>). If it is necessary to define <eszet> and <s><s> as an equivalence class, then a collating element must be defined for the string ss.

All characters specified via an ellipsis will by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit UNDEFINED special symbol will by default be assigned the same primary weight (that is, belong to the same equivalence class). An ellipsis symbol as a weight is interpreted to mean that each character in the sequence has unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight is treated as an error if the collating element is neither an ellipsis nor the special symbol UNDEFINED.

The special keyword IGNORE as a weight indicates that when strings are compared using the weights at the level where IGNORE is specified, the collating element is ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to IGNORE in their primary weight form an equivalence class.

An empty operand is interpreted as the collating element itself.

For example, the order statement:

```
<a>      <a>;<a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and is interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section defines the interpretation of bracket expressions in regular expressions.

Example:

```
order_start                             forward;backward
UNDEFINED                               IGNORE;IGNORE
<LOW>
<space>                                 <LOW>;<space>
. . .                                   <LOW>;. . .
<a>                                     <a>;<a>
<a-acute>                               <a>;<a-acute>
<a-grave>                               <a>;<a-grave>
<A>                                     <a>;<A>
<A-acute>                               <a>;<A-acute>
<A-grave>                               <a>;<A-grave>
<ch>                                    <ch>;<ch>
<Ch>                                    <ch>;<Ch>
<s>                                     <s>;<s>
<eszet>                                 "<s><s>";"<eszet><eszet>"
order_end
```

This example is interpreted as follows:

1. The `UNDEFINED` means that all characters not specified in this definition (explicitly or via the ellipsis) are ignored for collation purposes; for regular expression purposes they are ordered first.
2. All characters between <`space`> and <`a`> have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
3. All characters based on the upper– or lower–case character `a` belong to the same primary equivalence class.
4. The multi-character collating element <`ch`> is represented by the collating symbol <`ch`> and belongs to the same primary equivalence class as the multi-character collating element <`Ch`>.

order_end *keyword* The collating order entries must be terminated with an `order_end` keyword.

LC_MONETARY The LC_MONETARY category defines the rules and symbols that are used to format monetary numeric information. This information is available through the localeconv(3C) function

The following items are defined in this category of the locale. The item names are the keywords recognized by the localedef(1) utility when defining a locale. They are also similar to the member names of the lconv structure defined in <locale.h>. The localeconv function returns {CHAR_MAX} for unspecified integer items and the empty string ("") for unspecified or size zero string items.

In a locale definition file the operands are strings. For some keywords, the strings can contain only integers. Keywords that are not provided, string values set to the empty string (""), or integer keywords set to –1, are used to indicate that the value is not available in the locale.

int_curr_symbol The international currency symbol. The operand is a four-character string, with the first three characters containing the alphabetic international currency symbol in accordance with those specified in the ISO 4217 standard. The fourth character is the character used to separate the international currency symbol from the monetary quantity.

currency_symbol The string used as the local currency symbol.

mon_decimal_point The operand is a string containing the symbol that is used as the decimal delimiter (radix character) in monetary formatted quantities.

mon_thousands_sep The operand is a string containing the symbol that is used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.

mon_grouping Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not –1, then the size of the previous group (if any) will be repeatedly used for the remainder of the digits. If the last integer is –1, then no further grouping will be performed.

The following is an example of the interpretation of the mon_grouping keyword. Assuming that the value to be formatted is 123456789 and the mon_thousands_sep is ', then the following table shows the result. The third column shows the equivalent string in the ISO C standard that would be used by the localeconv function to accommodate this grouping.

```
mon_grouping   Formatted Value  ISO C String

3;-1           123456’789       "\3\177"
3              123’456’789      "\3"
3;2;-1         1234’56’789      "\3\2\177"
3;2            12’34’56’789     "\3\2"
-1             1234567898       "\177"
```

In these examples, the octal value of {CHAR_MAX} is 177.

`positive_sign` A string used to indicate a non-negative-valued formatted monetary quantity.

`negative_sign` A string used to indicate a negative-valued formatted monetary quantity.

`int_frac_digits` An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using `int_curr_symbol`.

`frac_digits` An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using `currency_symbol`.

`p_cs_precedes` In an application conforming to the SUSv3 standard, an integer set to `1` if the `currency_symbol` precedes the value for a monetary quantity with a non-negative value, and set to `0` if the symbol succeeds the value.

In an application *not* conforming to the SUSv3 standard, an integer set to `1` if the `currency_symbol` or `int_currency_symbol` precedes the value for a monetary quantity with a non-negative value, and set to `0` if the symbol succeeds the value.

`p_sep_by_space` In an application conforming to the SUSv3 standard, an integer set to `0` if no space separates the `currency_symbol` from the value for a monetary quantity with a non-negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

In an application *not* conforming to the SUSv3 standard, an integer set to `0` if no space separates the `currency_symbol` or `int_curr_symbol` from the value for a monetary quantity with a non-negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

`n_cs_precedes`

In an application conforming to the SUSv3 standard, an integer set to `1` if the `currency_symbol` precedes the value for a monetary quantity with a negative value, and set to `0` if the symbol succeeds the value.

In an application *not* conforming to the SUSv3 standard, an integer set to `1` if the `currency_symbol` or `int_currency_symbol` precedes the value for a monetary quantity with a negative value, and set to `0` if the symbol succeeds the value.

`n_sep_by_space`

In an application conforming to the SUSv3 standard, an integer set to `0` if no space separates the `currency_symbol` from the value for a monetary quantity with a negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

In an application *not* conforming to the SUSv3 standard, an integer set to `0` if no space separates the `currency_symbol` or `int_curr_symbol` from the value for a monetary quantity with a negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

`p_sign_posn`

An integer set to a value indicating the positioning of the `positive_sign` for a monetary quantity with a non-negative value. The following integer values are recognized for both `p_sign_posn` and `n_sign_posn`:

In an application conforming to the SUSv3 standard:

`0` Parentheses enclose the quantity and the `currency_symbol`.

`1` The sign string precedes the quantity and the `currency_symbol`.

`2` The sign string succeeds the quantity and the `currency_symbol`.

`3` The sign string precedes the `currency_symbol`.

`4` The sign string succeeds the `currency_symbol`.

In an application *not* conforming to the SUSv3 standard:

`0` Parentheses enclose the quantity and the `currency_symbol` or `int_curr_symbol`.

`1` The sign string precedes the quantity and the `currency_symbol` or `int_curr_symbol`.

2 The sign string succeeds the quantity and the `currency_symbol` or `int_curr_symbol`.

3 The sign string precedes the `currency_symbol` or `int_curr_symbol`.

4 The sign string succeeds the `currency_symbol` or `int_curr_symbol`.

`n_sign_posn` An integer set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity.

`int_p_cs_precedes` An integer set to `1` if the `int_curr_symbol` precedes the value for a monetary quantity with a non-negative value, and set to `0` if the symbol succeeds the value.

`int_n_cs_precedes` An integer set to `1` if the `int_curr_symbol` precedes the value for a monetary quantity with a negative value, and set to `0` if the symbol succeeds the value.

`int_p_sep_by_space` An integer set to `0` if no space separates the `int_curr_symbol` from the value for a monetary quantity with a non-negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

`int_n_sep_by_space` An integer set to `0` if no space separates the `int_curr_symbol` from the value for a monetary quantity with a negative value, set to `1` if a space separates the symbol from the value, and set to `2` if a space separates the symbol and the sign string, if adjacent.

`int_p_sign_posn` An integer set to a value indicating the positioning of the `positive_sign` for a positive monetary quantity formatted with the international format. The following integer values are recognized for `int_p_sign_posn` and `int_n_sign_posn`:

`0` Parentheses enclose the quantity and the `int_curr_symbol`.

`1` The sign string precedes the quantity and the `int_curr_symbol`.

`2` The sign string precedes the quantity and the `int_curr_symbol`.

`3` The sign string precedes the `int_curr_symbol`.

`4` The sign string succeeds the `int_curr_symbol`.

`int_n_sign_posn` An integer set to a value indicating the positioning of the `negative_sign` for a negative monetary quantity formatted with the international format.

The following table shows the result of various combinations:

		p_sep_by_space		
		2	1	0
p_cs_precedes= 1	p_sign_posn= 0	($1.25)	($1.25)	($1.25)
	p_sign_posn= 1	+$1.25	+$1.25	+$1.25
	p_sign_posn= 2	$1.25+	$1.25+	$1.25+
	p_sign_posn= 3	+$1.25	+$1.25	+$1.25
	p_sign_posn= 4	$+1.25	$+1.25	$+1.25
p_cs_precedes= 0	p_sign_posn= 0	(1.25 $)	(1.25 $)	(1.25$)
	p_sign_posn= 1	+1.25 $	+1.25 $	+1.25$
	p_sign_posn= 2	1.25$ +	1.25 $+	1.25$+
	p_sign_posn= 3	1.25+ $	1.25 +$	1.25+$
	p_sign_posn= 4	1.25$ +	1.25 $+	1.25$+

The monetary formatting definitions for the POSIX locale follow. The code listing depicts the `localedef(1)` input, the table representing the same information with the addition of `localeconv(3C)` and `nl_langinfo(3C)` formats. All values are unspecified in the POSIX locale.

```
LC_MONETARY
# This is the POSIX locale definition for
# the LC_MONETARY category.
#
int_curr_symbol       ""
currency_symbol       ""
mon_decimal_point     ""
mon_thousands_sep     ""
mon_grouping          -1
positive_sign         ""
negative_sign         ""
int_frac_digits       -1
frac_digits           -1
p_cs_precedes         -1
p_sep_by_space        -1
n_cs_precedes         -1
n_sep_by_space        -1
p_sign_posn           -1
n_sign_posn           -1
int_p_cs_precedes     -1
```

```
int_p_sep_by_space    -1
int_n_cs_precedes     -1
int_n_sep_by_space    -1
int_p_sign_posn       -1
int_n_sign_posn       -1
#
END LC_MONETARY
```

The entry `n/a` indicates that the value is not available in the POSIX locale.

LC_NUMERIC The `LC_NUMERIC` category defines the rules and symbols that will be used to format non-monetary numeric information. This information is available through the `localeconv(3C)` function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the `localedef` utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in <`locale.h`>. The `localeconv()` function returns `{CHAR_MAX}` for unspecified integer items and the empty string (`""`) for unspecified or size zero string items.

In a locale definition file the operands are strings. For some keywords, the strings only can contain integers. Keywords that are not provided, string values set to the empty string (`""`), or integer keywords set to –1, will be used to indicate that the value is not available in the locale. The following keywords are recognized:

`decimal_point` The operand is a string containing the symbol that is used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the `decimal_point` to a single byte, the result of specifying a multi-byte operand is unspecified.

`thousands_sep` The operand is a string containing the symbol that is used as a separator for groups of digits to the left of the decimal delimiter in numeric, non-monetary formatted monetary quantities. In contexts where standards limit the `thousands_sep` to a single byte, the result of specifying a multi-byte operand is unspecified.

`grouping` Define the size of each group of digits in formatted non-monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not –1, then the size of the previous group (if any) will be repeatedly used for the remainder of the digits. If the last integer is –1, then no further grouping will be performed. The non-monetary numeric formatting definitions for the POSIX locale follow. The code listing depicts the `localedef` input, the table representing the same information with the addition of `localeconv` values, and `nl_langinfo` constants.

```
LC_NUMERIC
# This is the POSIX locale definition for
# the LC_NUMERIC category.
#
decimal_point    "<period>"
thousands_sep    ""
grouping    -1
#
END LC_NUMERIC
```

	POSIX locale	langinfo	localeconv()	localedef
Item	Value	Constant	Value	Value
decimal_point	"."	RADIXCHAR	"."	.
thousands_sep	n/a	THOUSEP	""	""
grouping	n/a	-	""	–1

The entry `n/a` indicates that the value is not available in the POSIX locale.

LC_TIME The `LC_TIME` category defines the interpretation of the field descriptors supported by `date(1)` and affects the behavior of the `strftime(3C)`, `wcsftime(3C)`, `strptime(3C)`, and `nl_langinfo(3C)` functions. Because the interfaces for C-language access and locale definition differ significantly, they are described separately. For locale definition, the following mandatory keywords are recognized:

`abday` Define the abbreviated weekday names, corresponding to the `%a` field descriptor (conversion specification in the `strftime()`, `wcsftime()`, and `strptime()` functions). The operand consists of seven semicolon-separated strings, each surrounded by double-quotes. The first string is the abbreviated name of the day corresponding to Sunday, the second the abbreviated name of the day corresponding to Monday, and so on.

`day` Define the full weekday names, corresponding to the `%A` field descriptor. The operand consists of seven semicolon-separated strings, each surrounded by double-quotes. The first string is the full name of the day corresponding to Sunday, the second the full name of the day corresponding to Monday, and so on.

`abmon` Define the abbreviated month names, corresponding to the `%b` field descriptor. The operand consists of twelve semicolon-separated strings, each surrounded by double-quotes. The first string is the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.

`mon` Define the full month names, corresponding to the `%B` field descriptor. The operand consists of twelve semicolon-separated strings, each surrounded by double-quotes. The first string is the full name of the first month of the year (January), the second the full name of the second month, and so on.

`d_t_fmt` Define the appropriate date and time representation, corresponding to the `%c` field descriptor. The operand consists of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain the escape sequences \\, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

`date_fmt` Define the appropriate date and time representation, corresponding to the `%C` field descriptor. The operand consists of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain the escape sequences `\\`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

`d_fmt` Define the appropriate date representation, corresponding to the `%x` field descriptor. The operand consists of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain the escape sequences `\\`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

`t_fmt` Define the appropriate time representation, corresponding to the `%X` field descriptor. The operand consists of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain the escape sequences `\\`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

`am_pm` Define the appropriate representation of the *ante meridiem* and *post meridiem* strings, corresponding to the `%p` field descriptor. The operand consists of two strings, separated by a semicolon, each surrounded by double-quotes. The first string represents the *ante meridiem* designation, the last string the *post meridiem* designation.

`t_fmt_ampm` Define the appropriate time representation in the 12-hour clock format with `am_pm`, corresponding to the `%r` field descriptor. The operand consists of a string and can contain any combination of characters and field descriptors. If the string is empty, the 12-hour format is not supported in the locale.

`era` Define how years are counted and displayed for each era in a locale. The operand consists of semicolon-separated strings. Each string is an era description segment with the format:

direction:*offset*:*start_date*:*end_date*:*era_name*:*era_format*

according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.

The start of an era might not be the earliest point For example, the Christian era B.C. starts on the day before January 1, A.D. 1, and increases with earlier time.

direction Either a + or a – character. The + character indicates that years closer to the *start_date* have lower numbers than those closer to the *end_date*. The – character indicates that years closer to the *start_date* have higher numbers than those closer to the *end_date*.

offset The number of the year closest to the *start_date* in the era, corresponding to the `%Eg` and `%Ey` field descriptors.

start_date A date in the form *yyyy*/*mm*/`dd`, where *yyyy*, *mm*, and `dd` are the year, month and day numbers respectively of the start of the era. Years prior to A.D. 1 are represented as negative numbers.

end_date The ending date of the era, in the same format as the *start_date*, or one of the two special values –* or +*. The value –* indicates that the ending date is the beginning of time. The value +* indicates that the ending date is the end of time.

era_name A string representing the name of the era, corresponding to the `%EC` field descriptor.

era_format A string for formatting the year in the era, corresponding to the `%EG` and `%EY` field descriptors.

`era_d_fmt` Define the format of the date in alternative era notation, corresponding to the `%Ex` field descriptor.

`era_t_fmt` Define the locale's appropriate alternative time format, corresponding to the `%EX` field descriptor.

`era_d_t_fmt` Define the locale's appropriate alternative date and time format, corresponding to the `%Ec` field descriptor.

`alt_digits` Define alternative symbols for digits, corresponding to the `%O` field descriptor modifier. The operand consists of semicolon-separated strings, each surrounded by double-quotes. The first string is the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The `%O` modifier indicates that the string corresponding to the value specified via the field descriptor will be used instead of the value.

LC_TIME *C-language* Access

The following information can be accessed. These correspond to constants defined in `<langinfo.h>` and used as arguments to the `nl_langinfo(3C)` function.

`ABDAY_`*x* The abbreviated weekday names (for example Sun), where *x* is a number from 1 to 7.

`DAY_`*x* The full weekday names (for example Sunday), where *x* is a number from 1 to 7.

`ABMON_`*x* The abbreviated month names (for example Jan), where *x* is a number from 1 to 12.

`MON_`*x* The full month names (for example January), where *x* is a number from 1 to 12.

`D_T_FMT` The appropriate date and time representation.

`D_FMT` The appropriate date representation.

`T_FMT` The appropriate time representation.

`AM_STR` The appropriate ante-meridiem affix.

`PM_STR` The appropriate post-meridiem affix.

`T_FMT_AMPM` The appropriate time representation in the 12-hour clock format with `AM_STR` and `PM_STR`.

`ERA` The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment has the format:

direction:*offset*:*start_date*:*end_date*:*era_name*:*era_format*

according to the definitions below. There will be as many era description segments as are necessary to describe the different eras. Era description segments are separated by semicolons.

The start of an era might not be the earliest point For example, the Christian era B.C. starts on the day before January 1, A.D. 1, and increases with earlier time.

direction Either a + or a – character. The + character indicates that years closer to the *start_date* have lower numbers than those closer to the *end_date*. The – character indicates that years closer to the *start_date* have higher numbers than those closer to the *end_date*.

offset The number of the year closest to the start_date in the era.

start_date A date in the form *yyyy*/*mm*/*dd*, where *yyyy*, *mm*, and `dd` are the year, month and day numbers respectively of the start of the era. Years prior to AD 1 are represented as negative numbers.

end_date The ending date of the era, in the same format as the *start_date*, or one of the two special values, –* or +*. The

value –* indicates that the ending date is the beginning of time. The value +* indicates that the ending date is the end of time.

era_name The era, corresponding to the %EC conversion specification.

era_format The format of the year in the era, corresponding to the %EY and %EY conversion specifications.

ERA_D_FMT The era date format.

ERA_T_FMT The locale's appropriate alternative time format, corresponding to the %EX field descriptor.

ERA_D_T_FMT The locale's appropriate alternative date and time format, corresponding to the %Ec field descriptor.

ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion specification modifier. The value consists of semicolon-separated symbols. The first is the alternative symbol corresponding to zero, the second is the symbol corresponding to one, and so on. Up to 100 alternative symbols may be specified. The following table displays the correspondence between the items described above and the conversion specifiers used by date(1) and the strftime(3C), wcsftime(3C), and strptime(3C) functions.

localedef Keyword	langinfo Constant	Conversion Specifier
abday	ABDAY_*x*	%a
day	DAY_*x*	%A
abmon	ABMON_*x*	%b
mon	MON	%B
d_t_fmt	D_T_FMT	%c
date_fmt	DATE_FMT	%C
d_fmt	D_FMT	%x
t_fmt	T_FMT	%X
am_pm	AM_STR	%p
am_pm	PM_STR	%p
t_fmt_ampm	T_FMT_AMPM	%r
era	ERA	%EC, %Eg,

localedef Keyword	langinfo Constant	Conversion Specifier
		%EG, %Ey, %EY
era_d_fmt	ERA_D_FMT	%Ex
era_t_fmt	ERA_T_FMT	%EX
era_d_t_fmt	ERA_D_T_FMT	%Ec
alt_digits	ALT_DIGITS	%O

LC_TIME *General Information*

Although certain of the field descriptors in the POSIX locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The LC_TIME descriptions of abday, day, mon, and abmon imply a Gregorian style calendar (7-day weeks, 12-month years, leap years, and so forth). Formatting time strings for other types of calendars is outside the scope of this document set.

As specified under date in Locale Definition and strftime(3C), the field descriptors corresponding to the optional keywords consist of a modifier followed by a traditional field descriptor (for instance %Ex). If the optional keywords are not supported by the implementation or are unspecified for the current locale, these field descriptors are treated as the traditional field descriptor. For instance, assume the following keywords:

```
alt_digits    "0th" ; "1st" ; "2nd" ; "3rd" ; "4th" ; "5th" ; \
"6th" ; "7th" ; "8th" ; "9th" ; "10th">
d_fmt    "The %Od day of %B in %Y"
```

On 7/4/1776, the %x field descriptor would result in “The 4th day of July in 1776” while 7/14/1789 would come out as “The 14 day of July in 1789” The above example is for illustrative purposes only. The %O modifier is primarily intended to provide for Kanji or Hindi digits in date formats.

LC_MESSAGES

The LC_MESSAGES category defines the format and values for affirmative and negative responses.

The following keywords are recognized as part of the locale definition file. The nl_langinfo(3C) function accepts upper-case versions of the first four keywords.

yesexpr — The operand consists of an extended regular expression (see regex(5)) that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

noexpr — The operand consists of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

`yesstr` The operand consists of a fixed string (not a regular expression) that can be used by an application for composition of a message that lists an acceptable affirmative response, such as in a prompt.

`nostr` The operand consists of a fixed string that can be used by an application for composition of a message that lists an acceptable negative response. The format and values for affirmative and negative responses of the POSIX locale follow; the code listing depicting the `localedef` input, the table representing the same information with the addition of `nl_langinfo()` constants.

```
LC_MESSAGES
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y>\
        <right-square-bracket>"
#
noexpr  "<circumflex><left-square-bracket><n><N>\
        <right-square-bracket>"
#
yesstr    "yes"
nostr    "no"
END LC_MESSAGES
```

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YESEXPR	"^[yY]"
noexpr	NOEXPR	"^[nN]"
yesstr	YESSTR	"yes"
nostr	NOSTR	"no"

In an application conforming to the SUSv3 standard, the information on `yesstr` and `nostr` is not available.

See Also `date(1)`, `locale(1)`, `localedef(1)`, `sort(1)`, `tr(1)`, `uniq(1)`, `localeconv(3C)`, `nl_langinfo(3C)`, `setlocale(3C)`, `strcoll(3C)`, `strftime(3C)`, `strptime(3C)`, `strxfrm(3C)`, `wcscoll(3C)`, `wcsftime(3C)`, `wcsxfrm(3C)`, `wctype(3C)`, `attributes(5)`, `charmap(5)`, `extensions(5)`, `regex(5)`

Name lx – Linux branded zone

Description The `lx` brand uses the branded zones framework described in `brands(5)` to enable Linux binary applications to run unmodified on a machine with a Solaris Operating System kernel.

The `lx` brand includes the tools necessary to install a CentOS 3.x or Red Hat Enterprise Linux 3.x distribution inside a non-global zone. The brand supports the execution of 32-bit Linux applications on x86/x64 machines running the Solaris system in either 32-bit or 64-bit mode.

Supported Linux Distributions

The `lx` brand emulates the system call interfaces provided by the Linux 2.4.21 kernel, as modified by Red Hat in the RHEL 3.x distributions. This kernel provides the system call interfaces consumed by the `glibc` version 2.3.2 released by Red Hat.

In addition, the `lx` brand partially emulates the Linux `/dev` and `/proc` interfaces.

Configuration and Administration

The `lx` brand supports the whole root non-global zone model. You cannot use the `inherit-pkg-dir` resource with the `lx` brand. All of the required linux packages are installed into the private file systems of the zone.

The `zonecfg(1M)` utility is used to configure an `lx` branded zone. Once a branded zone has been installed, that zone's brand cannot be changed or removed. The `zoneadm(1M)` utility is used to report the zone's brand type and administer the zone. The `zlogin(1)` utility is used to log in to the zone.

Application Support

The `lx` zone only supports user-level Linux applications. You cannot use Linux device drivers, Linux kernel modules, or Linux file systems from inside an `lx` zone.

You cannot add any non-standard Solaris devices to a Linux zone. Any attempt to do so will result in a zone that `zonecfg(1M)` will refuse to verify.

You cannot run Solaris applications inside an `lx` zone. Solaris debugging tools such as `DTrace` (see `dtrace(1M)`) and `mdb` (see `mdb(1)`) can be applied to Linux processes executing inside the zone, but the tools themselves must be running in the global zone. Any core files generated are produced in the Solaris format, and such files can only be debugged with Solaris tools.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWlxr, SUNWlxu
Interface Stability	Evolving

See Also `mdb(1)`, `zlogin(1)`, `zonename(1)`, `dtrace(1M)`, `zoneadm(1M)`, `zonecfg(1M)`, `brands(5)`, `zones(5)`, `lx_systrace(7D)`

Name man – macros to format Reference Manual pages

Synopsis `nroff -man` *filename*`...`

`troff -man` *filename*`...`

Description These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by the `man(1)` command. See the "Conventions" section.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way `.I` may be used to italicize a whole line, or `.SB` may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by `-man`:

`\*R` '®', '(Reg)' in `nroff`.

`\*S` Change to default type size.

Requests * n.t.l. = next text line; p.i. = prevailing indent

Request	*Cause*	*If no*	*Explanation*
	Break	*Argument*	
`.B` *t*	no	*t*=n.t.l.*	Text is in bold font.
`.BI` *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
`.BR` *t*	no	*t*=n.t.l.	Join words, alternating bold and roman.
`.DT`	no	.5i 1i...	Restore default tabs.
`.HP` *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
`.I` *t*	no	*t*=n.t.l.	Text is italic.
`.IB` *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
`.IP` *x i*	yes	*x*=""	Same as `.TP` with tag *x*.
`.IR` *t*	no	*t*=n.t.l.	Join words, alternating italic and roman.

Request	*Cause Break*	*If no Argument*	*Explanation*
`.IX` *t*	no	-	Index macro, for SunSoft internal use.
`.LP`	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
`.P`	yes	-	Same as `.LP`.
`.PD` *d*	no	*d*=.4v	Set vertical distance between paragraphs.
`.PP`	yes	-	Same as `.LP`.
`.RE`	yes	-	End of relative indent. Restores prevailing indent.
`.RB` *t*	no	*t*=n.t.l.	Join words, alternating roman and bold.
`.RI` *t*	no	*t*=n.t.l.	Join words, alternating roman and italic.
`.RS` *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
`.SB` *t*	no	-	Reduce size of text by 1 point, make text bold.
`.SH` *t*	yes	-	Section Heading.
`.SM` *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
`.SS` *t*	yes	*t*=n.t.l.	Section Subheading.
`.TH` *n s d f m*	yes	-	Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
`.TP` *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
`.TX` *t p*	no	-	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a manual page, `man` examines the first line to determine whether it requires special processing. For example a first line consisting of:

```
'\" t
```

indicates that the manual page must be run through the `tbl(1)` preprocessor.

A typical manual page for a command or function is laid out as follows:

.TH *title* [1-9]

The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no `troff(1)` commands or escapes, and no macro requests. It is used to generate the `windex` database, which is used by the `whatis(1)` command.

.SH SYNOPSIS

Commands: The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

- [ ] An argument, when surrounded by brackets is optional.
- | Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.
- . . . Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions: If required, the data declaration, or `#include` directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

	Literal text from the synopsis appears in constant width, as do literal filenames and references to items that appear elsewhere in the reference manuals. Arguments are italicized. If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a `USAGE` section, which follows the `OPTIONS` section. The `DESCRIPTION` section only describes the behavior of the command itself, not that of subcommands.
.SH OPTIONS	The list of options along with a description of how each affects the command's operation.
.SH RETURN VALUES	A list of the values the library routine will return to the calling program and the conditions that cause these values to be returned.
.SH EXIT STATUS	A list of the values the utility will return to the calling program or shell, and the conditions that cause these values to be returned.
.SH FILES	A list of files associated with the command or function.
.SH SEE ALSO	A comma-separated list of related manual pages, followed by references to other published materials.
.SH DIAGNOSTICS	A list of diagnostic messages and an explanation of each.
.SH BUGS	A description of limitations, known defects, and possible problems associated with the command or function.

Files `/usr/share/lib/tmac/an`

`/usr/share/man/windex`

See Also `man(1)`, `nroff(1)`, `troff(1)`, `whatis(1)`

Dale Dougherty and Tim O'Reilly, *Unix Text Processing*

Name mansun – macros to format Reference Manual pages

Synopsis

```
nroff  -mansun filename...
troff  -mansun filename...
```

Description These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by `man(1)`. See the "Conventions" section.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way `.I` may be used to italicize a whole line, or `.SB` may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by `-mansun`:

`\*R` '®', '(Reg)' in `nroff`.

`\*S` Change to default type size.

Requests * n.t.l. = next text line; p.i. = prevailing indent

Request	*Cause*	*If no*	*Explanation*
	Break	*Argument*	
`.B` *t*	no	*t*=n.t.l.*	Text is in bold font.
`.BI` *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
`.BR` *t*	no	*t*=n.t.l.	Join words, alternating bold and Roman.
`.DT`	no	.5i 1i...	Restore default tabs.
`.HP` *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
`.I` *t*	no	*t*=n.t.l.	Text is italic.
`.IB` *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
`.IP` *x i*	yes	*x*=""	Same as `.TP` with tag *x*.

Request	*Cause Break*	*If no Argument*	*Explanation*
`.IR` *t*	no	*t*=n.t.l.	Join words, alternating italic and Roman.
`.IX` *t*	no	-	Index macro, for SunSoft internal use.
`.LP`	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
`.P`	yes	-	Same as `.LP`.
`.PD` *d*	no	*d*=.4v	Set vertical distance between paragraphs.
`.PP`	yes	-	Same as `.LP`.
`.RE`	yes	-	End of relative indent. Restores prevailing indent.
`.RB` *t*	no	*t*=n.t.l.	Join words, alternating Roman and bold.
`.RI` *t*	no	*t*=n.t.l.	Join words, alternating Roman and italic.
`.RS` *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
`.SB` *t*	no	-	Reduce size of text by 1 point, make text bold.
`.SH` *t*	yes	-	Section Heading.
`.SM` *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
`.SS` *t*	yes	*t*=n.t.l.	Section Subheading.
`.TH` *n s d f m*	yes	-	Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
`.TP` *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
`.TX` *t p*	no	-	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a manual page, `mansun` examines the first line to determine whether it requires special processing. For example a first line consisting of:

```
'\" t
```

indicates that the manual page must be run through the `tbl(1)` preprocessor.

A typical manual page for a command or function is laid out as follows:

.TH *title* [1-8] The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in Roman font, this section contains no `troff(1)` commands or escapes, and no macro requests. It is used to generate the `windex` database, which is used by the `whatis(1)` command.

.SH SYNOPSIS

Commands: The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in Roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.

. . . Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions: If required, the data declaration, or `#include` directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in constant width, as do literal filenames and references to items that appear elsewhere in the reference manuals. Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a `USAGE`

	section, which follows the `OPTIONS` section. The `DESCRIPTION` section only describes the behavior of the command itself, not that of subcommands.
.SH OPTIONS	The list of options along with a description of how each affects the command's operation.
.SH FILES	A list of files associated with the command or function.
.SH SEE ALSO	A comma-separated list of related manual pages, followed by references to other published materials.
.SH DIAGNOSTICS	A list of diagnostic messages and an explanation of each.
.SH BUGS	A description of limitations, known defects, and possible problems associated with the command or function.

Files `/usr/share/lib/tmac/ansun`

`/usr/share/man/windex`

See Also `man(1)`, `nroff(1)`, `troff(1)`, `whatis(1)`

Dale Dougherty and Tim O'Reilly, *Unix Text Processing*

Name me – macros for formatting papers

Synopsis

```
nroff  -me [options] filename...
troff  -me [options] filename...
```

Description This package of `nroff` and `troff` macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through `col(1)`.

The macro requests are defined below. Many `nroff` and `troff` requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first `.pp`:

.bp	begin new page
.br	break output line here
.sp *n*	insert *n* spacing lines
.ls *n*	(line spacing) *n*=1 single, *n*=2 double space
.na	no alignment of right margin
.ce *n*	center next *n* lines
.ul *n*	underline next *n* lines
.sz +*n*	add *n* to point size

Output of the `eqn(1)`, `neqn(1)`, `refer(1)`, and `tbl(1)` preprocessors for equations and tables is acceptable as input.

Requests In the following list, "initialization" refers to the first `.pp`, `.lp`, `.ip`, `.np`, `.sh`, or `.uh` macro. This list is incomplete.

Request	*Initial Value*	*Cause Break*	*Explanation*
`.(c`	-	yes	Begin centered block.
`.(d`	-	no	Begin delayed text.
`.(f`	-	no	Begin footnote.
`.(l`	-	yes	Begin list.
`.(q`	-	yes	Begin major quote.
`.(x`*x*	-	no	Begin indexed item in index *x*.
`.(z`	-	no	Begin floating keep.

Request	*Initial Value*	*Cause Break*	*Explanation*
.)c	-	yes	End centered block.
.)d	-	yes	End delayed text.
.)f	-	yes	End footnote.
.)l	-	yes	End list.
.)q	-	yes	End major quote.
.)x	-	yes	End index item.
.)z	-	yes	End floating keep.
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be C (chapter), A (appendix), P (preliminary, for instance, abstract, table of contents, etc.), B (bibliography), RC (chapters renumbered from page one each chapter), or RA (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by eqn or neqn.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or

Request	*Initial Value*	*Cause Break*	*Explanation*
			C to center the equation.
`.GE`	-	yes	End *gremlin* picture.
`.GS`	-	yes	Begin *gremlin* picture.
`.PE`	-	yes	End `pic` picture.
`.PS`	-	yes	Begin `pic` picture.
`.TE`	-	yes	End table.
`.TH`	-	yes	End heading section of table.
`.TS` *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
`.ac` *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
`.b` *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
`.ba` +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
`.bc`	no	yes	Begin new column.
`.bi` *x*	no	no	Print *x* in bold italics (nofill only).
`.bu`	-	yes	Begin bulleted paragraph.
`.bx` *x*	no	no	Print *x* in a box (nofill only).
`.ef` *'x'y'z*	"""	no	Set even footer to *x y z*.
`.eh` *'x'y'z*	"""	no	Set even header to *x y z*.
`.fo` *'x'y'z*	"""	no	Set footer to *x y z*.
`.hx`	-	no	Suppress headers and footers on

Request	*Initial Value*	*Cause Break*	*Explanation*
			next page.
`.he` *'x'y'z*	"""	no	Set header to *x y z*.
`.hl`	-	yes	Draw a horizontal line.
`.i` *x*	no	no	Italicize *x*; if *x* missing, italic text follows.
`.ip` *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
`.lp`	yes	yes	Start left-blocked paragraph.
`.lo`	-	no	Read in a file of local macros of the form .**x*. Must be given before initialization.
`.np`	1	yes	Start numbered paragraph.
`.of` *'x'y'z*	"""	no	Set odd footer to x y z.
`.oh` *'x'y'z*	"""	no	Set odd header to x y z.
`.pd`	-	yes	Print delayed text.
`.pp`	no	yes	Begin paragraph. First line indented.
`.r`	yes	no	Roman text follows.
`.re`	-	no	Reset tabs to default values.
`.sc`	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
`.sh` *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
`.sk`	no	no	Leave the next page blank. Only one page is remembered ahead.

Request	*Initial Value*	*Cause Break*	*Explanation*
`.sm` *x*	-	no	Set *x* in a smaller pointsize.
`.sz` +*n*	10p	no	Augment the point size by *n* points.
`.th`	no	no	Produce the paper in thesis format. Must be given before initialization.
`.tp`	no	yes	Begin title page.
`.u` *x*	-	no	Underline argument (even in `troff`). (Nofill only).
`.uh`	-	yes	Like `.sh` but unnumbered.
`.xp` *x*	-	no	Print index *x*.

Files `/usr/share/lib/tmac/e`

`/usr/share/lib/tmac/*.me`

See Also `col`(1), `eqn`(1), `nroff`(1), `refer`(1), `tbl`(1), `troff`(1)

Name mech_spnego – Simple and Protected GSS-API Negotiation Mechanism

Synopsis `/usr/lib/gss/mech_spnego.so.1`

Description The SPNEGO security mechanism for GSS-API allows GSS-API applications to negotiate the actual security mechanism to be used in the GSS-API session. `mech_spnego.so.1` is a shared object module that is dynamically opened by applications that specify the SPNEGO Object Identifier (OID) in calls to the GSS-API functions (see `libgss(3LIB)`).

SPNEGO is described by IETF RFC 2478 and is intended to be used in environments where multiple GSS-API mechanisms are available to the client or server and neither side knows what mechanisms are supported by the other.

When SPNEGO is used, it selects the list of mechanisms to advertise by reading the GSS mechanism configuration file, `/etc/gss/mech` (see `mech(4)`), and by listing all active mechanisms except for itself.

Options SPNEGO may be configured to function in two ways. The first way is to interoperate with Microsoft SSPI clients and servers that use the Microsoft "`Negotiate`" method, which is also based on SPNEGO. The Microsoft "`Negotiate`" mechanism does not strictly follow the IETF RFC. Therefore, use special handling in order to enable full interoperability. In order to interoperate, place option "`[ msinterop ]`" at the end of the SPNEGO line in `/etc/gss/mech`.

This is an example (from `/etc/gss/mech`):

```
spnego    1.3.6.1.5.5.2    mech_spnego.so    [ msinterop ]
```

Without the "`[ msinterop ]`" option, `mech_spnego` will follow the strict IETF RFC 2478 specification and will not be able to negotiate with Microsoft applications that try to use the SSPI "`Negotiate`" mechanism.

Interfaces `mech_spnego.so.1` has no public interfaces. It is only activated and used through the GSS-API interface provided by `libgss.so.1` (see `libgss(3LIB)`).

Files

`/usr/lib/gss/mech_spnego.so.1`	shared object file
`/usr/lib/sparcv9/gss/mech_spnego.so.1`	SPARC 64-bit shared object file
`/usr/lib/amd64/gss/mech_spnego.so.1`	x86 64-bit shared object file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUWNspnego
MT Level	Safe

See Also `Intro`(3), `libgss`(3LIB), `mech`(4), `attributes`(5)

Solaris Security for Developers Guide

Name mm – text formatting (memorandum) macros

Synopsis

```
nroff  -mm [options] filename...

troff  -mm [options] filename...
```

Description This package of nroff(1) and troff(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through col(1). All external -mm macros are defined below.

Note: this -mm macro package is an extended version written at Berkeley and is a superset of the standard -mm macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many nroff and troff requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp*n*	insert n spacing lines
.ce*n*	center next n lines
.ls*n*	line spacing: *n*=1 single, *n*=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, \fIword\fR will italicize *word*. Output of the tbl(1), eqn(1) and refer(1) preprocessors for equations, tables, and references is acceptable as input.

Requests Here is a table of macros.

Macro Name	Initial Value	Break? Reset?	Explanation
.1C	on	y,y	one column format on a new page
.2C [*l*]	–	y,y	two column format *l*=line length
.AE	–	y	end abstract
.AL [*t*] [*i*] [*s*]	*t*=1;*i*=.Li;*s*=0	y	Start automatic list type *t*=[1,A,a,I,i] 1=arabic numbers; A=uppercase letters a=lowercase letters; I=uppercase Roman numerals; i=lowercase Roman numerals indentation *i*; separation *s*

Macro Name	Initial Value	Break? Reset?	Explanation
.AS *m* [*n*]	*n*=0	y	begin abstract
.AU	–	y	author's name
.AV *x*	–	y	signature and date line of verifier *x*
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.BE	–	y	end block text
.BI *x y*	–	n	embolden *x* and underline *y*
.BL	–	y	bullet list
.BR *x y*	–	n	embolden *x* and use Roman font for *y*
.BS	–	n	start block text
.CN	–	y	same as .DE (nroff)
.CS	–	y	cover sheet
.CW	–	n	same as .DS I (nroff)
.DE	–	y	end display
.DF [*p*] [*f*] [*rp*]	*p*=L;*f*=N	y	start floating display; position *p*=[L,C,CB] L=left; I=indent; C=center; CB=center block fill *f*=[N,Y]; right position *rp* (fill only)
.DL [*i*] [*s*]	–	y	start dash list
.DS [*p*] [*f*] [*rp*]	*p*=L;*f*=N	y	begin static display (see .DF for argument descriptions)
.EC *x* [*n*]	*n*=1	y	equation title; equation *x*; number *n*
.EF *x*	–	n	even footer appears at the bottom of even-numbered pages; *x*="*l*'*c*'*r*" *l*=left; *c*=center; *r*=right
.EH *x*	–	n	even header appears at the top of even-numbered pages; *x*="*l*'*c*'*r*" *l*=left; *c*=center; *r*=right
.EN	–	y	end displayed equation produced by eqn
.EQ	–	y	break out equation produced by eqn
.EX *x* [*n*]	*n*=1	y	exhibit title; exhibit *x*
			number *n*

Macro Name	Initial Value	Break? Reset?	Explanation
.FD [*f*] [*r*]	*f*=0;*r*=1	n	set footnote style format *f*=[0-11]; renumber *r*=[0,1]
.FE	–	y	end footnote
.FG *x* [*n*]	*n*=1	y	figure title; figure *x*; number *n*
.FS	–	n	start footnote
.H *l* [*t*]	–	y	produce numbered heading level *l*=[1-7]; title *t*
.HU *t*	–	y	produce unnumbered heading; title *t*
.I *x*	–	n	underline *x*
.IB *x y*	–	n	underline *x* and embolden *y*
.IR *x y*	–	n	underline *x* and use Roman font on *y*
.LE [*s*]	*s*=0	y	end list; separation *s*
.LI [*m*] [*p*]	–	y	start new list item; mark *m*
			prefix *p* (mark only)
.ML *m* [*i*] [*s*]	*s*=0	y	start marked list; mark *m* indentation *i*; separation *s*=[0,1]
.MT *x*		y	memo title; title *x*
.ND *x*		n	no date in page footer; *x* is date on cover
.NE	–	y	end block text
.NS	–	y	start block text
.OF *x*	–	n	odd footer appears at the bottom of odd-numbered pages; *x*="*l* ' *c* ' *r*" *l*=left; *c*=center; *r*=right
.OF *x*	–	n	odd header appears at the top of odd-numbered pages; *x*="*l* ' *c* ' *r*" *l*=left; *c*=center; *r*=right
.OP	–	y	skip to the top of an odd-number page
.P [*t*]	*t*=0	y,y	begin paragraph; *t*=[0,1] 0=justified; 1=indented
.PF *x*	–	n	page footer appears at the bottom of every page; *x*="*l* ' *c* ' *r*" *l*=left; *c*=center; *r*=right

Macro Name	Initial Value	Break? Reset?	Explanation
.PH *x*	–	n	page header appears at the top of every page; *x*="*l' c' r*" *l*=left; *c*=center; *r*=right
.R	on	n	return to Roman font
.RB *x y*	–	n	use Roman on *x* and embolden *y*
.RI *x y*	–	n	use Roman on *x* and underline *y*
.RP *x*	-	y,y	released paper format ? *x*=no stops title on first
.RS	5n	y,y	right shift: start level of relative indentation
.S *m n*	–	n	set character point size & vertical space character point size *m*; vertical space *n*
.SA *x*	*x*=1	n	justification; *x*=[0,1]
.SK *x*	–	y	skip *x* pages
.SM	–	n	smaller; decrease point size by 2
.SP [*x*]	–	y	leave *x* blank lines
.TB *x* [*n*]	*n*=1	y	table title; table *x*; number *n*
.TC	–	y	print table of contents (put at end of input file)
.TE	–	y	end of table processed by tbl
.TH	–	y	end multi-page header of table
.TL	–	n	title in boldface and two points larger
.TM	–	n	UC Berkeley thesis mode
.TP *i*	y	y	*i*=p.i. Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.TY	–	y	display centered title CONTENTS
.VL *i* [*m*] [*s*]	*m*=0;*s*=0	y	start variable-item list; indentation *i* mark-indentation *m*; separation *s*

Registers Formatting distances can be controlled in -mm by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
Cl	contents level	table of contents	2
De	display eject	display	0
Df	display floating	display	5
Ds	display spacing	display	1v
Hb	heading break	heading	2
Hc	heading centering	heading	0
Hi	heading indent	heading	1
Hi	heading spacing	heading	1
Hu	heading unnumbered	heading	2
Li	list indentation	list	6 (nroff) 5 (troff)
Ls	list spacing	list	6
Pi	paragraph indent	paragraph	5
Pt	paragraph type	paragraph	1
Si	static indent	display	5 (nroff) 3 (troff)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting Pi to 0 suppresses paragraph indentation

Here is a list of string registers available in -mm; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in nroff, “ in troff)
*U	unquote (" in nroff, ’’ in troff)
*–	dash (-- in nroff, — in troff)
*(MO	month (month of the year)

Name	String's Function
*(DY	day (current date)
**	automatically numbered footnote
*'	acute accent (before letter)
*‘	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)
\(BU	bullet item
\(DT	date (*month day*, *yr*)
\(EM	em dash
\(Lf	LIST OF FIGURES title
\(Lt	LIST OF TABLES title
\(Lx	LIST OF EXHIBITS title
\(Le	LIST OF EQUATIONS title
\(Rp	REFERENCES title
\(Tm	trademark character (TM)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

Files /usr/share/lib/tmac/m

/usr/share/lib/tmac/mm.[nt] nroff and troff definitions of mm.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdoc

See Also col(1), eqn(1), nroff(1), refer(1), tbl(1), troff(1), attributes(5)

Bugs Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

Name mms – MMS service overview

Description MMS is a distributed removable media management system. It is based on the standard, *IEEE 1244, Media Management System (MMS)*.

The current implementation of MMS manages tapes and cartridges in a Sun StorageTek ACSLS tape library and emulated tapes on a file system. MMS is capable of sharing drives among multiple hosts. An important feature of MMS is that it provides a uniform and consistent tape interface to its client applications.

MMS consists of a media manager (MM) which manages the entire MMS system. Information about the system is maintained by MM in a database.

Every library managed by MMS has an active library manager (LM). An LM provides library access and mounting service to MMS.

A drive managed by MMS has a drive manager (DM) on each host that has access to the drive. A DM provides device specific operations. A DM also provides label processing. The current implementation supports only ANSI labels.

MMS client applications request MM to mount and unmount cartridges and process them after they are mounted. Applications use the Media Management Protocol (MMP) to make requests. A library of client application functions is available.

The Solaris implementation of the MMS uses the `mmsinit(1M)` command to configure and start the MMS service and the `mmsadm(1M)` command to list and set parameters required by the service.

The MMS service instances, as reported by `svcs(1)`, are:

`mms:wcr`
watcher

`mms:db`
Media Manager PostgreSQL database

`mms:mm`
Media Manager

The MMS service RBAC (see `rbac(5)`) authorizations are:

- `solaris.smf.manage.mms`
- `solaris.smf.value.mms`

The following are the MMS service configuration data accessible through the service management facility (see `smf(5)`):

`config/type`
Host is media manager or watcher only. Valid values are `server` or `client`.

`manager/host`
Host name where the media manager is running.

`manager/port`
Port the media manager is using.

`ssl/enabled`
Media Manager is using SSL. Valid values are `true` or `false`. Note the Media Manager needs an SSL RSA certificate and Diffie-Hellman parameters must be 'true'.

`ssl/verify`
Media Manager-only value, determines whether two-way authentication is enabled, which requires all Media Manager clients to have SSL RSA certificate. Valid values are are `true` for two-way authentication and `false` for one-way authentication.

`ssl/cert_file`
Name of file that contains the SSL RSA certificate, private key, and RSA certificate chain.

`ssl/pass_file`
Name of file for the private key password phrase if the RSA certificate is encrypted.

`ssl/dh_file`
Media Manager-only value, name of file that stores the Diffie-Hellman parameters.

`ssl/crl_file`
Name of optional CRL file.

`ssl/cipher`
Optional cipher change from `EDH-RSA-DES-CBC3-SHA`.

`ssl/peer_file`
Optional client only, name of file that stores the media manager certificate. The certificate is checked against the certificate returned by `SSL_get_peer_certificate()` function.

The following are the MMS Database fault management resource identifier (FMRI) configuration data:

`postgresql/bin`
Directory where the PostgreSQL binaries are located. The directory is specific to the version of PostgreSQL qualified for use by the MMS software.

`postgresql/data`
Directory where the PostgreSQL database is located. The default location is a version-specific subdirectory of `/var/mms/db`.

The following are the MMS Media Manager FMRI configuration data:

`db/host`
For future use. Set to `localhost`.

`db/port`
Port number to use for PostgreSQL login. The default value is `7656`.

`db/user`
PostgreSQL user name. The default user name is `postgres`.

`db/name`
Name of PostgreSQL database to which to connect. The default value is `mms`.

`option/trace`
Optional. The Media Manager trace level before connecting to the PostgreSQL database.

`option/db_reconnect_max_retry`
Number of times to wait for the database to accept a Media Manager connection.

`option/db_reconnect_timeout`
Number of seconds to wait between reconnect tries.

The following are the MMS Watcher FMRI configuration data:

`option/ssi_path`
Path to ACSLS SSI daemon.

`option/libapi_path`
Path to ACSLS API library.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

See Also `svcs(1)`, `mmsadm(1M)`, `mmsclient(1M)`, `mmsexplorer(1M)`, `mmsinit(1M)`, `mmsclient_script(4)`, `attributes(5)`, `rbac(5)`, `smf(5)`

Name ms – text formatting macros

Synopsis

```
nroff  -ms [options] filename...

troff  -ms [options] filename...
```

Description This package of nroff(1) and troff(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through col(1). All external -ms macros are defined below.

Note: this -ms macro package is an extended version written at Berkeley and is a superset of the standard -ms macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many nroff and troff requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp *n*	insert n spacing lines
.ce *n*	center next n lines
.ls *n*	line spacing: *n*=1 single, *n*=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, \fIword\fR will italicize *word*. Output of the tbl(1), eqn(1) and refer(1) preprocessors for equations, tables, and references is acceptable as input.

Requests

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no do not label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box

Macro Name	Initial Value	Break? Reset?	Explanation
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I, L, C, B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by eqn
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.

Macro Name	Initial Value	Break? Reset?	Explanation
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on first page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on first page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set TAB characters to 8n 16n . . . (nroff) or 5n 10n . . . (troff)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by tbl
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in troff
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; y=indent

Macro Name	Initial Value	Break? Reset?	Explanation
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line indented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.] –	–	n	beginning of refer reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

Registers Formatting distances can be controlled in -ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ≈1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in -ms; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in nroff, “ in troff)
*U	unquote (" in nroff, ” in troff)
*–	dash (-- in nroff, – in troff)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*’	acute accent (before letter)
*‘	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

Files /usr/share/lib/tmac/s

/usr/share/lib/tmac/ms.???

See Also col(1), eqn(1), nroff(1), refer(1), tbl(1), troff(1)

Bugs Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

Name mutex – concepts relating to mutual exclusion locks

Description Mutual exclusion locks (mutexes) prevent multiple threads from simultaneously executing critical sections of code which access shared data (that is, mutexes are used to serialize the execution of threads). All mutexes must be global. A successful call to acquire a mutex will cause another thread that is also trying to lock the same mutex to block until the owner thread unlocks the mutex.

Mutexes can synchronize threads within the same process or in other processes. Mutexes can be used to synchronize threads between processes if the mutexes are allocated in writable memory and shared among the cooperating processes (see `mmap(2)`), and have been initialized for this task.

The following table lists mutex functions and the actions they perform.

FUNCTION	ACTION
`mutex_init`	Initialize a mutex.
`mutex_destroy`	Destroy a mutex.
`mutex_lock`	Lock a mutex.
`mutex_trylock`	Attempt to lock a mutex.
`mutex_unlock`	Unlock a mutex.
`pthread_mutex_init`	Initialize a mutex.
`pthread_mutex_destroy`	Destroy a mutex.
`pthread_mutex_lock`	Lock a mutex.
`pthread_mutex_trylock`	Attempt to lock a mutex.
`pthread_mutex_unlock`	Unlock a mutex.

Initialization Mutexes are either intra-process or inter-process, depending upon the argument passed implicitly or explicitly to the initialization of that mutex. A statically allocated mutex does not need to be explicitly initialized; by default, a statically allocated mutex is initialized with all zeros and its scope is set to be within the calling process.

For inter-process synchronization, a mutex needs to be allocated in memory shared between these processes. Since the memory for such a mutex must be allocated dynamically, the mutex needs to be explicitly initialized with the appropriate attribute that indicates inter-process use.

Locking and Unlocking A critical section of code is enclosed by a call to lock the mutex and the call to unlock the mutex to protect it from simultaneous access by multiple threads. Only one thread at a time may possess mutually exclusive access to the critical section of code that is enclosed by the mutex-locking call and the mutex-unlocking call, whether the mutex's scope is intra-process

or inter-process. A thread calling to lock the mutex either gets exclusive access to the code starting from the successful locking until its call to unlock the mutex, or it waits until the mutex is unlocked by the thread that locked it.

Mutexes have ownership, unlike semaphores. Only the thread that locked a mutex, (that is, the owner of the mutex), should unlock it.

If a thread waiting for a mutex receives a signal, upon return from the signal handler, the thread resumes waiting for the mutex as if there was no interrupt.

Caveats

Mutexes are almost like data – they can be embedded in data structures, files, dynamic or static memory, and so forth. Hence, they are easy to introduce into a program. However, too many mutexes can degrade performance and scalability of the application. Because too few mutexes can hinder the concurrency of the application, they should be introduced with care. Also, incorrect usage (such as recursive calls, or violation of locking order, and so forth) can lead to deadlocks, or worse, data inconsistencies.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also

mmap(2), shmop(2), mutex_destroy(3C), mutex_init(3C), mutex_lock(3C), mutex_trylock(3C), mutex_unlock(3C), pthread_create(3C), pthread_mutex_destroy(3C), pthread_mutex_init(3C), pthread_mutex_lock(3C), pthread_mutex_trylock(3C), pthread_mutex_unlock(3C), pthread_mutexattr_init(3C), attributes(5), standards(5)

Notes

In the current implementation of threads, `pthread_mutex_lock()`, `pthread_mutex_unlock()`, `mutex_lock()` `mutex_unlock()`, `pthread_mutex_trylock()`, and `mutex_trylock()` do not validate the mutex type. Therefore, an uninitialized mutex or a mutex with an invalid type does not return `EINVAL`. Interfaces for mutexes with an invalid type have unspecified behavior.

By default, if multiple threads are waiting for a mutex, the order of acquisition is undefined.

The system does not support multiple mappings to the same logical synch object if it is initialized as process-private (`USYNC_THREAD` for Solaris, `PTHREAD_PROCESS_PRIVATE` for POSIX). If you need to mmap(2)a synch object to different locations within the same address space, then the synch object should be initialized as a shared object (`USYNC_PROCESS` for Solaris, `PTHREAD_PROCESS_SHARED` for POSIX).

Name native – native branded zone

Description The native brand uses the branded zones framework described in brands(5) to run zones installed with the same software as is installed in the global zone. The system software must always be in sync with the global zone when using a native brand.

Sub-commands The following native brand-specific subcommand options are supported in zoneadm(1M).

`attach` [`-u`] [`-b` *patchid*]...

For native zones, `zoneadm` checks package and patch levels on the machine to which the zone is to be attached. If the packages/patches that the zone depends on from the global zone are different (have different revision numbers) from the dependent packages/patches on the source machine, `zoneadm` reports these conflicts and does not perform the attach. If the destination system has only newer dependent packages/patches (higher revision numbers) than those on the source system, you can use the `-u` option to update the attached zone to match the `-revision` packages and patches that exist on the new system. With `-u`, as in the default behavior, `zoneadm` does not perform an attach if outdated packages/patches are found on the target system.

For native zones, one or more `-b` options can be used to specify a patch ID for a patch installed in the zone. These patches will be backed out before the zone is attached or, if `-u` was also specified, updated.

`install` [`-a` *archive*] | [`-d` *path*] [`-p`] [`-s`] [`-u`] [`-v`] [`-b` *patchid*]...

The native brand installer supports installing the zone from either the software already installed on the system or from an image of an installed system running the same release. This can be a full flash archive (see flash_archive(4)) or a cpio(1) or pax(1) "xustar" archive. The `cpio` archive be compressed with `gzip` or `bzip2`. The image can also be a level `0` ufsdump(1M), a path to the top-level of a system's root tree, or a pre-existing zone path.

With no options the zone is installed using same software as is running the global zone.

To install the zone from a system image either the `-a` or `-d` is required. Either the `-u` or `-p` option is also required in this case.

`-a` *archive*

The path to a flash_archive(4), cpio(1), or pax(1) "xustar" archive, or a level`0` ufsdump(1M), of an installed system. `cpio` archives may be compressed using `gzip` or `bzip2`.

`-b` *patchid*

One or more `-b` options can be used to specify a patch ID for a patch installed in the system image. These patches will be backed out during the installation process.

`-d` *path*

The path to the root directory of an installed system. If path is a hyphen (`-`), the *zonepath* is presumed to be already populated with the system image.

`-p`
Preserve the system configuration after installing the zone.

`-s`
Install silently.

`-u`
Run `sys-unconfig(1M)` on the zone after installing it.

`-v`
Verbose output from the install process.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWzoneu
Interface Stability	Uncommitted

See Also `cpio(1)`, `pax(1)`, `zlogin(1)`, `zonename(1)`, `sys-unconfig(1M)`, `ufsdump(1M)`, `zoneadm(1M)`, `zonecfg(1M)`, `flash_archive(4)`, `attributes(5)`, `brands(5)`, `privileges(5)`, `zones(5)`, `lx_systrace(7D)`

Name nfssec – overview of NFS security modes

Description The mount_nfs(1M) and share_nfs(1M) commands each provide a way to specify the security mode to be used on an NFS file system through the sec=*mode* option. *mode* can be sys, dh, krb5, krb5i, krb5p, or none. These security modes can also be added to the automount maps. Note that mount_nfs(1M) and automount(1M) do not support sec=none at this time. mount_nfs(1M) allows you to specify a single security mode; share_nfs(1M) allows you to specify multiple modes (or none). With multiple modes, an NFS client can choose any of the modes in the list.

The sec=*mode* option on the share_nfs(1M) command line establishes the security mode ofNFS servers. If the NFS connection uses the NFS Version 3 protocol, the NFS clients must query the server for the appropriate *mode* to use. If the NFS connection uses the NFS Version 2 protocol, then the NFS client uses the default security mode, which is currently sys. NFS clients may force the use of a specific security mode by specifying the sec=*mode* option on the command line. However, if the file system on the server is not shared with that security mode, the client may be denied access.

If the NFS client wants to authenticate the NFS server using a particular (stronger) security mode, the client wants to specify the security mode to be used, even if the connection uses the NFS Version 3 protocol. This guarantees that an attacker masquerading as the server does not compromise the client.

The NFS security modes are described below. Of these, the krb5, krb5i, krb5p modes use the Kerberos V5 protocol for authenticating and protecting the shared filesystems. Before these can be used, the system must be configured to be part of a Kerberos realm. See kerberos(5).

sys — Use AUTH_SYS authentication. The user's UNIX user-id and group-ids are passed in the clear on the network, unauthenticated by the NFS server. This is the simplest security method and requires no additional administration. It is the default used by Solaris NFS Version 2 clients and Solaris NFS servers.

dh — Use a Diffie-Hellman public key system (AUTH_DES, which is referred to as AUTH_DH in *RFC 2695: Authentication Mechanisms for ONC RPC*.

krb5 — Use Kerberos V5 protocol to authenticate users before granting access to the shared filesystem.

krb5i — Use Kerberos V5 authentication with integrity checking (checksums) to verify that the data has not been tampered with.

krb5p — User Kerberos V5 authentication, integrity checksums, and privacy protection (encryption) on the shared filesystem. This provides the most secure filesystem sharing, as all traffic is encrypted. It should be noted that performance might suffer on some systems when using krb5p, depending on the computational intensity of the encryption algorithm and the amount of data being transferred.

`none` Use null authentication (`AUTH_NONE`). NFS clients using `AUTH_NONE` have no identity and are mapped to the anonymous user `nobody` by NFS servers. A client using a security mode other than the one with which a Solaris NFS server shares the file system has its security mode mapped to `AUTH_NONE`. In this case, if the file system is shared with `sec=`*none*, users from the client are mapped to the anonymous user. The NFS security mode `none` is supported by `share_nfs`(1M), but not by `mount_nfs`(1M) or `automount`(1M).

Files `/etc/nfssec.conf` NFS security service configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWnfscr

See Also `automount`(1M), `kclient`(1M), `mount_nfs`(1M), `share_nfs`(1M), `rpc_clnt_auth`(3NSL), `secure_rpc`(3NSL), `nfssec.conf`(4), `attributes`(5), `kerberos`(5)

RFC 2695: Authentication Mechanisms for ONC RPC

Notes `/etc/nfssec.conf` lists the NFS security services. Do not edit this file. It is not intended to be user-configurable. See `kclient`(1M).

Name nis, NIS, NIS+ – network information name service

Description NIS, formerly known as the Yellow Pages or YP, is the name of the network information name service in common use in networks on which Sun and other vendors' network nodes reside. The most recent version of NIS is version 2.

All commands and functions that use NIS version 2 are prefixed by the letters yp as in `ypmatch(1)`, `ypcat(1)`, `yp_match(3NSL)`, and `yp_first(3NSL)`.

The NIS+ name service is no longer shipped with the Solaris operating system. Tools to aid the migration from NIS+ to LDAP are available in the current Solaris release. For more information, visit http://www.sun.com/directory/nisplus/transition.html.

Name openssl – OpenSSL cryptographic and Secure Sockets Layer toolkit

Description OpenSSL is a cryptography toolkit that implements the Secure Sockets Layer (SSLv2/v3) and Transport Layer Security (TLS v1) network protocols.

The following features are omitted from the binaries for issues including but not limited to patents, trademark, and US export restrictions: ECC, IDEA, MDC2, RC3, RC5, Dynamic Engine Loading, 4758_CCA Engine, AEP Engine, Atalla Engine, CHIL Engine, CSWIFT Engine, GMP Engine, NURON Engine, PadLock Engine, Sureware Engine, and UBSEC Engine.

A new PKCS#11 engine has been included with ENGINE name `pkcs11`. The engine was developed in Sun and is not integrated in the OpenSSL project.

The PKCS#11 engine is configured to use the Solaris Cryptographic Framework. See `cryptoadm(1M)` for configuration information.

The PKCS#11 engine can support the following set of mechanisms: CKM_AES_CBC, CKM_AES_ECB, CKM_BLOWFISH_CBC, CKM_DES_CBC, CKM_DES_ECB, CKM_DES3_CBC, CKM_DES3_ECB, CKM_DSA, CKM_MD5, CKM_RC4, CKM_RSA_PKCS, CKM_RSA_X_509, CKM_SHA_1, CKM_SHA224, CKM_SHA256, CKM_SHA384, and CKM_SHA512.

The set of mechanisms available depends on installed Crypto Framework providers. To see what mechanisms can be offloaded to the Cryptographic Framework through the PKCS#11 engine on a given machine, run the following command:

```
/usr/sfw/bin/openssl engine -vvv -t -c
```

Due to requirements of the PKCS#11 standard regarding `fork(2)` behavior, some applications that use the OpenSSL EVP interfaces and `fork()` with active crypto contexts might experience unexpected behavior.

Using FIPS Mode A FIPS Capable OpenSSL is available in `/lib/openssl/fips-140`. To use this version of OpenSSL on a per-application basis, `LD_LIBRARY_PATH` can be set. `crle(1)` can be used to select this version of OpenSSL for all applications.

Example:

```
# crle -a /lib/libcrypto.so.0.9.8 -o \
   /lib/openssl/fips-140
# crle -64 -a /lib/64/libcrypto.so.0.9.8 -o \
   /lib/openssl/fips-140/64
```

The FIPS Capable `libcrypto` and the non-FIPS Capable `libcrypto` are ABI compatible. One exception to this is the use of the `CRYPTO_NUM_LOCKS` preprocessor macro. Instead of using `CRYPTO_NUM_LOCKS`, the `CRYPTO_num_locks(3openssl)` function should be used instead.

Even when a FIPS Capable OpenSSL is used applications cannot automatically claim FIPS compliance. See the *OpenSSL FIPS 140-2 User Guide* at http://openssl.org/ for more information.

`openssl(1openssl)` can be run in FIPS mode. The environmental variable, `OPENSSL_FIPS`, must be set and the FIPS Capable OpenSSL libraries must be used.

Example:

```
# export LD_LIBRARY_PATH=/lib/openssl/fips-140
# export OPENSSL_FIPS=1
# openssl version
OpenSSL 0.9.8k-fips 25 Mar 2009 (security fixes for: CVE-
2009-1377 CVE-2009-1378 CVE-2009-1379)
```

Building an OpenSSL Application

To build an OpenSSL application, use the following `cc` command line options:

`cc [ `*flag*`... ] `*file*`... -lcrypto -lssl [ `*library*`... ]`

To build an OpenSSL application which supports a FIPS mode of operation, use the following `cc` command line options:

```
cc -I/usr/include/openssl/fips-140 -L/lib/openssl/fips-140 \
   [ flag... ] file... -lcrypto -lssl [ library... ]
```

Accessing RSA Keys in PKCS#11 Keystores

OpenSSL can access RSA keys in PKCS#11 keystores using the following functions of the ENGINE API:

```
EVP_PKEY *ENGINE_load_private_key(ENGINE *e,
      const char *key_id, UI_METHOD *ui_method,
   void *callback_data)

EVP_PKEY *ENGINE_load_public_key(ENGINE *e,
       const char *key_id, UI_METHOD *ui_method,
   void *callback_data)
```

`key_id`, formerly for filenames only, can be now also set to a `PKCS#11 URI`. The `EVP_PKEY` structure is newly allocated and caller is responsible to free the structure later. To avoid clashes with existing filenames, `file://` prefix for filenames is now also accepted but only when the PKCS#11 engine is in use. The PKCS#11 URI specification follows:

```
pkcs11:[token=<label>][:manuf=<label>][;serial=<label>]
   [;model=<label>][;object=<label>]
   [;objecttype=(public|private|cert)]
   [;passphrasedialog=(builtin|exec:<file>)]
```

The ordering of keywords is not significant. The PKCS#11 engine uses the keystore for the slot chosen for public key operations, which is `metaslot` on a standard configured machine. Currently, the PKCS#11 engine ignores the `objecttype` keyword. The only mandatory keyword is `object` which is the key object label. For information on how to use a different, possibly hardware, keystore with `metaslot`. See`libpkcs11(3LIB)`.

The token PIN is provided by way of the `passphrasedialog` keyword and is either read from the terminal (`builtin`) or from the output of an external command (`exec:<file>`). The PIN is used to log into the token and by default is deleted from the memory then. The keyword `pin` is intentionally not provided due to inherent security problems of possible use of a password in the process arguments

Due to fork safety issues the application must re-login if the child continues to use the PKCS#11 engine. It is done inside of the engine automatically if fork is detected and in that case, `exec:<file>` option of the `passphrasedialog` keyword can be used. Alternatively, an environment variable `OPENSSL_PKCS11_PIN_CACHING_POLICY` can be used to allow the PIN to be cached in memory and reused in the child. It can be set to `none` which is the default, `memory` to store the PIN in memory, and `mlocked-memory` keep the PIN in a locked page using `mlock(3C)`. `PRIV_PROC_LOCK_MEMORY` privilege is required in that case.

Sensitive parts of private keys are never read from the token to the process memory no matter whether the key is tagged with sensitive flag or not. The PKCS#11 engine uses the public components as a search key to get a PKCS#11 object handle to the private key.

To use the RSA keys by reference, high level API functions such as `RSA_public_decrypt()()`, `EVP_PKEY_set1_RSA()()`, or `EVP_SignInit()()` must be used. Low level functions might go around the engine and fail to make use of the feature.

Additional Documentation

Extensive additional documentation for OpenSSL modules is available in the `/usr/share/man/man1openssl`, `/usr/share/man/man3openssl`, `/usr/share/man/man5openssl`, and `/usr/share/man/man7openssl` directories.

To view the license terms, attribution, and copyright for OpenSSL, see `/var/sadm/pkg/SUNWopensslr/install/copyright`.

Examples

EXAMPLE 1 Generating and Printing a Public Key

The following example generates and prints a public key stored in an already initilized PKCS#11 keystore. Notice the use of `-engine pkcs11` and `-inform e`.

```
$ pktool gencert keystore=pkcs11 label=mykey \
   subject="CN=test" keytype=rsa keylen=1024 serial=01
$ openssl rsa -in "pkcs11:object=mykey;passphrasedialog=builtin"\
   -pubout -text -engine pkcs11 -inform e
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWopensslr, SUNWopenssl
Interface Stability	Volatile

See Also crle(1), cryptoadm(1M), libpkcs11(3LIB), attributes(5), privileges(5)

/usr/share/man/man1openssl/openssl.1openssl, /usr/share/man/man1openssl/CRYPTO_num_locks.3openssl, /usr/share/man/man3openssl/engine.3, /usr/share/man/man3openssl/evp.3

OpenSSL FIPS 140-2 User Guide at http://openssl.org/

Name pam_allow – PAM authentication, account, session and password management PAM module to allow operations

Synopsis `pam_allow.so.1`

Description The `pam_allow` module implements all the PAM service module functions and returns `PAM_SUCCESS` for all calls. Opposite functionality is available in the `pam_deny(5)` module.

Proper Solaris authentication operation requires `pam_unix_cred(5)` be stacked above `pam_allow`.

The following options are interpreted:

debug Provides `syslog(3C)` debugging information at the `LOG_AUTH | LOG_DEBUG` level.

Errors `PAM_SUCCESS` is always returned.

Examples **EXAMPLE 1** Allowing `ssh none`

The following example is a `pam.conf` fragment that illustrates a sample for allowing `ssh none` authentication:

```
sshd-none  auth    required      pam_unix_cred.so.1
sshd-none  auth    sufficient    pam_allow.so.1
sshd-none  account sufficient    pam_allow.so.1
sshd-none  session sufficient    pam_allow.so.1
sshd-none  password sufficient   pam_allow.so.1
```

EXAMPLE 2 Allowing Kiosk Automatic Login Service

The following is example is a `pam.conf` fragment that illustrates a sample for allowing `gdm` kiosk auto login:

```
gdm-autologin auth  required    pam_unix_cred.so.1
gdm-autologin auth  sufficient  pam_allow.so.1
```

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

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

See Also `libpam(3LIB)`, `pam(3PAM)`, `pam_sm(3PAM)`, `syslog(3C)`, `pam.conf(4)`, `attributes(5)`, `pam_deny(5)`, `pam_unix_cred(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

This module is intended to be used to either allow access to specific services names, or to all service names not specified (by specifying it as the default service stack).

Name pam_authtok_check – authentication and password management module

Synopsis `pam_authtok_check.so.1`

Description `pam_authtok_check` provides functionality to the Password Management stack. The implementation of `pam_sm_chauthtok()` performs a number of checks on the construction of the newly entered password. `pam_sm_chauthtok()` is invoked twice by the PAM framework, once with flags set to `PAM_PRELIM_CHECK`, and once with flags set to `PAM_UPDATE_AUTHTOK`. This module only performs its checks during the first invocation. This module expects the current authentication token in the `PAM_OLDAUTHTOK` item, the new (to be checked) password in the `PAM_AUTHTOK` item, and the login name in the `PAM_USER` item. The checks performed by this module are:

length	The password length should not be less that the minimum specified in `/etc/default/passwd`.
circular shift	The password should not be a circular shift of the login name. This check may be disabled in `/etc/default/passwd`.
complexity	The password should contain at least the minimum number of characters described by the parameters `MINALPHA`, `MINNONALPHA`, `MINDIGIT`, and `MINSPECIAL`. Note that `MINNONALPHA` describes the same character classes as `MINDIGIT` and `MINSPECIAL` combined; therefore the user cannot specify both `MINNONALPHA` and `MINSPECIAL` (or `MINDIGIT`). The user must choose which of the two options to use. Furthermore, the `WHITESPACE` parameter determines whether whitespace characters are allowed. If unspecified `MINALPHA` is 2, `MINNONALPHA` is 1 and `WHITESPACE` is yes
variation	The old and new passwords must differ by at least the `MINDIFF` value specified in `/etc/default/passwd`. If unspecified, the default is 3. For accounts in name services which support password history checking, if prior history is defined, the new password must not match the prior passwords.
dictionary check	The password must not be based on a dictionary word. The list of words to be used for the site's dictionary can be specified with `DICTIONLIST`. It should contain a comma-separated list of filenames, one word per line. The database that is created from these files is stored in the directory named by `DICTIONDBDIR` (defaults to `/var/passwd`). See `mkpwdict(1M)` for information on pre-generating the database. If neither `DICTIONLIST` nor `DICTIONDBDIR` is specified, no dictionary check is made.
upper/lower case	The password must contain at least the minimum of upper- and lower-case letters specified by the `MINUPPER` and `MINLOWER` values in `/etc/default/passwd`. If unspecified, the defaults are 0.

maximum repeats — The password must not contain more consecutively repeating characters than specified by the MAXREPEATS value in /etc/default/passwd. If unspecified, no repeat character check is made.

The following option may be passed to the module:

force_check — If the PAM_NO_AUTHTOK_CHECK flag set, force_check ignores this flag. The PAM_NO_AUTHTOK_CHECK flag can be set to bypass password checks (see pam_chauthtok(3PAM)).

debug — syslog(3C) debugging information at the LOG_DEBUG level

Return Values If the password in PAM_AUTHTOK passes all tests, PAM_SUCCESS is returned. If any of the tests fail, PAM_AUTHTOK_ERR is returned.

Files /etc/default/passwd — See passwd(1) for a description of the contents.

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

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

See Also passwd(1), pam(3PAM), mkpwdict(1M), pam_chauthtok(3PAM), syslog(3C), libpam(3LIB), pam.conf(4), passwd(4), shadow(4), attributes(5), pam_authtok_get(5), pam_authtok_store(5), pam_dhkeys(5), pam_passwd_auth(5), pam_unix_account(5), pam_unix_auth(5), pam_unix_session(5)

Notes The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The pam_unix(5) module is no longer supported. 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_authtok_get – authentication and password management module

Synopsis `pam_authtok_get.so.1`

Description The `pam_authtok_get` service module provides password prompting funtionality to the PAM stack. It implements `pam_sm_authenticate()` and `pam_sm_chauthtok()`, providing functionality to both the Authentication Stack and the Password Management Stack.

Authentication Service

The implementation of `pam_sm_authenticate(3PAM)` prompts the user name if not set and then tries to get the authentication token from the pam handle. If the token is not set, it then prompts the user for a password and stores it in the `PAM` item `PAM_AUTHTOK`. This module is meant to be the first module on an authentication stack where users are to authenticate using a keyboard.

Password Management Service

Due to the nature of the PAM Password Management stack traversal mechanism, the `pam_sm_chauthtok(3PAM)` function is called twice. Once with the `PAM_PRELIM_CHECK` flag, and one with the `PAM_UPDATE_AUTHTOK` flag.

In the first (`PRELIM`) invocation, the implementation of `pam_sm_chauthtok(3PAM)` moves the contents of the `PAM_AUTHTOK` (current authentication token) to `PAM_OLDAUTHTOK`, and subsequentially prompts the user for a new password. This new password is stored in `PAM_AUTHTOK`.

If a previous module has set `PAM_OLDAUTHTOK` prior to the invocation of pam_authtok_get, this module turns into a `NO-OP` and immediately returns `PAM_SUCCESS`.

In the second (`UPDATE`) invocation, the user is prompted to Re-enter his password. The pam_sm_chauthtok implementation verifies this reentered password with the password stored in `PAM_AUTHTOK`. If the passwords match, the module returns `PAM_SUCCESS`.

The following option can be passed to the module:

debug `syslog(3C)` debugging information at the `LOG_DEBUG` level

Errors The authentication service returns the following error codes:

`PAM_SUCCESS`	Successfully obtains authentication token
`PAM_SYSTEM_ERR`	Fails to retrieve username, username is `NULL` or empty

The password management service returns the following error codes:

`PAM_SUCCESS`	Successfully obtains authentication token
`PAM_AUTHTOK_ERR`	Authentication token manipulation error

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

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

See Also pam(3PAM), pam_authenticate(3PAM), syslog(3C), libpam(3LIB), 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_account(5), pam_unix_auth(5), pam_unix_session(5)

Notes The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The pam_unix(5) module is no longer supported. 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_authtok_store – password management module

Synopsis `pam_authtok_store.so.1`

Description `pam_authtok_store` provides functionality to the PAM password management stack. It provides one function: `pam_sm_chauthtok()`.

When invoked with flags set to `PAM_UPDATE_AUTHTOK`, this module updates the authentication token for the user specified by `PAM_USER`.

The authentication token `PAM_OLDAUTHTOK` can be used to authenticate the user against repositories that need updating (NIS, LDAP). After successful updates, the new authentication token stored in `PAM_AUTHTOK` is the user's valid password.

This module honors the `PAM_REPOSITORY` item, which, if set, specifies which repository is to be updated. If `PAM_REPOSITORY` is unset, it follows the `nsswitch.conf(4)`.

The following option can be passed to the module:

`debug`	`syslog(3C)` debugging information at the `LOG_DEBUG` level
`server_policy`	If the account authority for the user, as specified by `PAM_USER`, is a server, do not encrypt the authentication token before updating.

Errors

`PAM_SUCCESS`	Successfully obtains authentication token
`PAM_SYSTEM_ERR`	Fails to get username, service name, old password or new password, user name null or empty, or password null.

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

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

See Also `pam(3PAM)`, `pam_authenticate(3PAM)`, `pam_chauthtok(3PAM)`, `syslog(3C)`, `libpam(3LIB)`, `pam.conf(4)`, `attributes(5)`, `pam_authtok_check(5)`, `pam_authtok_get(5)`, `pam_dhkeys(5)`, `pam_passwd_auth(5)`, `pam_unix_account(5)`, `pam_unix_auth(5)`, `pam_unix_session(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The `pam_unix(5)` module is no longer supported. 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)`.

If the `PAM_REPOSITORY` *item_type* is set and a service module does not recognize the type, the service module does not process any information, and returns `PAM_IGNORE`. If the `PAM_REPOSITORY` *item_type* is not set, a service module performs its default action.

Name pam_deny – PAM authentication, account, session and password management PAM module to deny operations

Synopsis

```
pam_deny.so.1
```

Description The `pam_deny` module implements all the PAM service module functions and returns the module type default failure return code for all calls.

The following options are interpreted:

debug `syslog(3C)` debugging information at the `LOG_AUTH|LOG_DEBUG` levels

Errors The following error codes are returned:

`PAM_ACCT_EXPIRED` If `pam_sm_acct_mgmt` is called.

`PAM_AUTH_ERR` If `pam_sm_authenticate` is called.

`PAM_AUTHOK_ERR` If `pam_sm_chauthtok` is called.

`PAM_CRED_ERR` If `pam_sm_setcred` is called.

`PAM_SESSION_ERR` If `pam_sm_open_session` or `pam_sm_close_session` is called.

Examples **EXAMPLE 1** Disallowing ssh none authentication

```
sshd-none       auth       requisite    pam_deny.so.1
sshd-none       account    requisite    pam_deny.so.1
sshd-none       session    requisite    pam_deny.so.1
sshd-none       password   requisite    pam_deny.so.1
```

EXAMPLE 2 Disallowing any service not explicitly defined

```
other           auth       requisite    pam_deny.so.1
other           account    requisite    pam_deny.so.1
other           session    requisite    pam_deny.so.1
other           password   requisite    pam_deny.so.1
```

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

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

See Also `su(1M)`, `libpam(3LIB)`, `pam(3PAM)`, `pam_sm_authenticate(3PAM)`, `syslog(3C)`, `pam.conf(4)`, `nsswitch.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_account(5)`, `pam_unix_auth(5)`, `pam_unix_session(5)`, `privileges(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The `pam_deny` module is intended to deny access to a specified service. The `other` service name may be used to deny access to services not explicitly specified.

Name pam_dhkeys – authentication Diffie-Hellman keys management module

Synopsis `pam_dhkeys.so.1`

Description The `pam_dhkeys.so.1` service module provides functionality to two PAM services: Secure RPC authentication and Secure RPC authentication token management.

Secure RPC authentication differs from regular Unix authentication because ONC RPCs use Secure RPC as the underlying security mechanism.

The following options may be passed to the module:

debug `syslog(3C)` debugging information at `LOG_DEBUG` level

nowarn Turn off warning messages

Authentication Services If the user has Diffie-Hellman keys, `pam_sm_authenticate()` establishes secret keys for the user specified by the `PAM_USER` (equivalent to running `keylogin(1)`), using the authentication token found in the `PAM_AUTHTOK` item. If `pam_sm_setcred()` is called with `PAM_ESTABLISH_CRED` and the user's secure RPC credentials need to be established, these credentials are set. This is equivalent to running `keylogin`(1).

If the credentials could not be set and `PAM_SILENT` is not specified, a diagnostic message is displayed. If `pam_setcred()` is called with `PAM_DELETE_CRED`, the user's secure RPC credentials are unset. This is equivalent to running `keylogout(1)`.

`PAM_REINITIALIZE_CRED` and `PAM_REFRESH_CRED` are not supported and return `PAM_IGNORE`.

Authentication Token Management The `pam_sm_chauthtok()` implementation checks whether the old login password decrypts the users secret keys. If it doesn't this module prompts the user for an old Secure RPC password and stores it in a pam data item called `SUNW_OLDRPCPASS`. This data item can be used by the store module to effectively update the users secret keys.

Errors The authentication service returns the following error codes:

`PAM_SUCCESS`	Credentials set successfully.
`PAM_IGNORE`	Credentials not needed to access the password repository.
`PAM_USER_UNKNOWN`	`PAM_USER` is not set, or the user is unknown.
`PAM_AUTH_ERR`	No secret keys were set. `PAM_AUTHTOK` is not set, no credentials are present or there is a wrong password.
`PAM_BUF_ERR`	Module ran out of memory.

The authentication token management returns the following error codes:

`PAM_SUCCESS`	Old `rpc` password is set in `SUNW_OLDRPCPASS`
`PAM_USER_UNKNOWN`	User in `PAM_USER` is unknown.

PAM_AUTHTOK_ERR	User did not provide a password that decrypts the secret keys.
PAM_BUF_ERR	Module ran out of memory.

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

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

See Also keylogin(1), keylogout(1), pam(3PAM), pam_authenticate(3PAM), pam_chauthtok(3PAM), pam_setcred(3PAM), pam_get_item(3PAM), pam_set_data(3PAM), pam_get_data(3PAM), syslog(3C), libpam(3LIB), pam.conf(4), attributes(5), pam_authtok_check(5), pam_authtok_get(5), pam_authtok_store(5), pam_passwd_auth(5), pam_unix_account(5), pam_unix_auth(5), pam_unix_session(5)

Notes The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The pam_unix(5) module is no longer supported. 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_dial_auth – authentication management PAM module for dialups

Synopsis `pam_dial_auth.so.1`

Description The `pam_dial_auth` module implements `pam_sm_authenticate(3PAM)` which authenticates the user according to the `dialups(4)` and `d_passwd(4)` files configuration.

Authentication service modules must implement both `pam_sm_authenticate()` and `pam_sm_setcred()`. `pam_sm_setcred()` in this module always returns `PAM_IGNORE`.

The value of the `PAM_TTY` item is checked against entries in `dialups(4)`. If there is a match, the user's shell is compared against entries in `d_passwd(4)`. If there is a matching entry, the user is prompted for a password which is validated against the entry found.

The following option may be passed in to this service module:

`debug` `syslog(3C)` debugging information at `LOG_DEBUG` level.

Errors If `dialups(4)` is not present, `PAM_IGNORE` is returned. Upon successful completion of `pam_sm_authenticate()`, `PAM_SUCCESS` is returned. The following error codes are returned upon error:

`PAM_AUTH_ERR`	Authentication failure.
`PAM_SERVICE_ERR`	Error in the calling service, `PAM_TTY` is not set.
`PAM_SYSTEM_ERR`	System error (`d_passwd(4)` is not present).
`PAM_USER_UNKNOWN`	No account is present for *user*.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

See Also `pam(3PAM)`, `pam_authenticate(3PAM)`, `pam_sm_authenticate(3PAM)`, `d_passwd(4)`, `dialups(4)`, `libpam(3LIB)`, `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_account(5)`, `pam_unix_auth(5)`, `pam_unix_session(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own `PAM` handle.

The `pam_unix(5)` module is no longer supported. 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_krb5 – authentication, account, session, and password management PAM modules for Kerberos V5

Synopsis `/usr/lib/security/pam_krb5.so.1`

Description The Kerberos V5 service module for PAM provides functionality for all four PAM modules: authentication, account management, session management, and password management. The service module is a shared object that can be dynamically loaded to provide the necessary functionality upon demand. Its path is specified in the PAM configuration file.

Kerberos Authentication Module

The Kerberos V5 authentication component provides functions to verify the identity of a user, `pam_sm_authenticate()`, and to manage the Kerberos credentials cache, `pam_sm_setcred()`.

`pam_sm_authenticate()` authenticates a user principal through the Kerberos authentication service. If the authentication request is successful, the authentication service sends a ticket-granting ticket (TGT) back to the service module, which then verifies that the TGT came from a valid Key Distribution Center (KDC) by attempting to get a service ticket for the local host service. For this to succeed, the local host's keytab file (`/etc/krb5/krb5.keytab`) must contain the entry for the local host service. For example, in the file `host/`*hostname.com*`@`*REALM*, *hostname.com* is the fully qualified local hostname and *REALM* is the default realm of the local host as defined in `/etc/krb5/krb5.conf`. If the host entry is not found in the keytab file, the authentication fails. Administrators may optionally disable this "strict" verification by setting "`verify_ap_req_nofail = false`" in `/etc/krb5/krb5.conf`. See `krb5.conf(4)` for more details on this option. This allows TGT verification to succeed in the absence of a keytab host principal entry.

`pam_sm_authenticate(3PAM)` may be passed the following flag:

`PAM_DISALLOW_NULL_AUTHTOK`
: This flag is ignored. The Kerberos authentication mechanism will not allow an empty password string by default.

`pam_sm_setcred()` creates and modifies the user's credential cache. This function initializes the user's credential cache, if it does not already exist, and stores the initial credentials for later use by Kerberized network applications. The following flags may be set in the flags field. They are best described by their effect on the user's credential cache.

`PAM_ESTABLISH_CRED`
: Stores the initial credentials in the user's credential cache so that the user may access Kerberos network services. If a successful authentication pass was made, the new credentials are stored in the credential cache, overwriting any existing credentials that were previously stored. If an unsuccessful authentication pass was made, PAM_CRED_UNAVAIL is returned.

PAM_DELETE_CRED
: This flag has no effect on the credential cache and always returns PAM_SUCCESS. The credential cache is not deleted because there is no accurate method to determine if the credentials are needed by another process. The credential cache may be deleted with the kdestroy(1) command.

PAM_REINITIALIZE_CRED
: Deletes the user's existing credential cache, if it exists, and creates a new credential cache. The new credentials are stored in the new cache and the user's ticket lifetime and renewable life time values are reset.

PAM_REFRESH_CRED
: Does not require a previous authentication pass, but if a successful one is made, the new credentials are stored in the credential cache. If a previous authentication pass was not made or was unsuccessful, an attempt to renew the existing credentials is made. Note that this function fails if the user's renewable ticket lifetime is expired.

The following options can be passed to the Kerberos V5 authentication module:

debug
: Provides syslog(3C) debugging information at LOG_DEBUG level.

nowarn
: Turns off warning messages.

Kerberos V5 Account Management Module

The Kerberos account management component provides a function to perform account management, pam_sm_acct_mgmt(). This function checks to see if the pam_krb5 authentication module has noted that the user's password has not expired. The following options may be passed in to the Kerberos V5 account management module:

debug
: Provides syslog(3C) debugging information at LOG_DEBUG level

nowarn
: Turns off warning messages. Also, does not query KDC for impending password expiration information used to warn the user.

Kerberos V5 Session Management Module

The Kerberos V5 session management component provides functions to initiate pam_sm_open_session() and terminate pam_sm_close_session() Kerberos sessions. For Kerberos V5, both pam_sm_open_session and pam_sm_close_session() are null functions, returning PAM_IGNORE.

Kerberos V5 Password Management Module

The Kerberos V5 password management component provides a function to change passwords, pam_sm_chauthtok(), in the Key Distribution Center (KDC) database. The following flags may be passed to pam_sm_chauthtok(3PAM):

PAM_CHANGE_EXPIRED_AUTHTOK
: The password service should only update the user's Kerberos password if it is expired. Otherwise, this function returns PAM_IGNORE. The default behaviour is to always change the user's Kerberos password.

PAM_PRELIM_CHECK
: This is a null function that always returns PAM_IGNORE.

PAM_UPDATE_AUTHTOK
This flag is necessary to change the user's Kerberos password. If this flag is not set, pam_krb5 returns PAM_SYSTEM_ERR.

The following option can be passed to the Kerberos V5 password module:

debug Provides syslog(3C) debugging information at LOG_DEBUG level.

Errors The following error codes are returned for pam_sm_authenticate():

PAM_AUTH_ERR	Authentication failure
PAM_BUF_ERR	Memory buffer error.
PAM_IGNORE	The user is "root" and the root key exists in the default keytab.
PAM_SUCCESS	Successfully obtained Kerberos credentials .
PAM_SYSTEM_ERR	System error.
PAM_USER_UNKNOWN	An unknown Kerberos principal was requested.

The following error codes are returned for pam_sm_setcred():

PAM_AUTH_ERR	Authentication failure.
PAM_BUF_ERR	Memory buffer error.
PAM_IGNORE	The user is "root" and the root key exists in the default keytab.
PAM_SYSTEM_ERR	System error.
PAM_SUCCESS	Successfully modified the Kerberos credential cache.

The following error codes are returned for pam_sm_acct_mgmt():

PAM_AUTH_ERR	Authentication failure.
PAM_IGNORE	Kerberos service module pam_sm_authenticate() was never called, or the user is "root" and the root key exists in the default keytab.
PAM_NEW_AUTHTOK_REQD	Obtain new authentication token from the user.
PAM_SERVICE_ERR	Error in underlying service module.
PAM_SUCCESS	Kerberos principal account is valid.
PAM_SYSTEM_ERR	System error.
PAM_USER_UNKNOWN	An unknown Kerberos principal was requested.

The following error code is returned for pam_sm_open_session() and pam_sm_close_session():

PAM_IGNORE These two functions are null functions in pam_krb5:

The following error codes are returned for pam_sm_chauthtok():

PAM_AUTH_ERR	Authentication failure.
PAM_IGNORE	The user has not been authenticated by Kerberos service module pam_sm_authenticate(), or the user is "root" and the root key exists in the default keytab.
PAM_NEW_AUTHTOK_REQD	User's Kerberos password has expired.
PAM_SERVICE_ERR	Error in module. At least one input parameter is missing.
PAM_SYSTEM_ERR	System error.
PAM_USER_UNKNOWN	An unknown Kerberos principal was requested.
PAM_SUCCESS	Successfully changed the user's Kerberos password.

Examples

EXAMPLE 1 Authenticate Users Through Kerberos as First Choice

The following is an excerpt of a sample pam.conf configuration file that authenticates users through the Kerberos authentication service and authenticates through the Unix login only if the Kerberos authentication fails. This arrangement is helpful when a majority of the users are networked by means of Kerberos and when there are only a few non-Kerberos type user accounts, such as root. The service illustrated below is for gdm.

```
gdm auth requisite          pam_authtok_get.so.1
gdm auth required           pam_dhkeys.so.1
gdm auth required           pam_unix_cred.so.1
gdm auth sufficient         pam_krb5.so.1
gdm auth required           pam_unix_auth.so.1
```

Note that these changes should not be made to the existing krlogin, krsh, and ktelnet service entries. Those services require Kerberos authentication, so using a seemingly sufficient control flag would not provide the necessary functionality for privacy and integrity. There should be no need to change those entries.

The following entries check for password expiration when dealing with Kerberos and Unix password aging policies:

```
other   account requisite       pam_roles.so.1
other   account required        pam_unix_account.so.1
other   account required        pam_krb5.so.1
```

The following entries would change the Kerberos password of the user and continue to change the Unix login password only if the Kerberos password change had failed:

```
other   password required       pam_dhkeys.so.1
other   password requisite      pam_authtok_get.so.1
```

EXAMPLE 1 Authenticate Users Through Kerberos as First Choice *(Continued)*

```
other   password requisite      pam_authtok_check.so.1
other   password sufficient     pam_krb5.so.1
other   password required       pam_authtok_store.so.1
```

When changing Kerberos based user's password, use `kpasswd(1)`. When changing a non-Kerberos user's password, it is recommended that the repository is specified (`-r`) with the `passwd(1)` command.

EXAMPLE 2 Authenticate Users Through Kerberos Only

The following example allows authentication only to users that have Kerberos-based accounts.

```
gdm auth requisite          pam_authtok_get.so.1
gdm auth required           pam_dhkeys.so.1
gdm auth required           pam_unix_cred.so.1
gdm auth binding            pam_krb5.so.1
gdm auth required           pam_unix_auth.so.1
```

Typically, you would have another service specified in the `pam.conf` file that would allow local users, such as database, web server, system administrator accounts, to log in to the host machine. For example, the service name “login” could be used for these users. Note that these users should not belong to any roles.

The rest of the module types look similar to that shown in the previous example:

```
other   account requisite       pam_roles.so.1
other   account required        pam_unix_account.so.1
other   account required        pam_krb5.so.1
```

With binding specified in the following, it is important that non-Kerberos users specify the repository in which they reside using the `-r` option with the `passwd(1)` command. This configuration is also based on the assumptions that:

- Kerberos users maintain only their Kerberos passwords;
- changing their Unix password is not necessary, given that they are authenticated only through their Kerberos passwords when logging in.

```
other   password required       pam_dhkeys.so.1
other   password requisite      pam_authtok_get.so.1
other   password requisite      pam_authtok_check.so.1
other   password binding        pam_krb5.so.1
other   password required       pam_authtok_store.so.1
```

EXAMPLE 3 Authenticate Through Kerberos Optionally

This configuration is helpful when the majority of users are non-Kerberos users and would like to authenticate through Kerberos if they happened to exist in the Kerberos database. The effect of this is similar to users voluntarily executing `kinit(1)` after they have successfully logged in:

```
gdm auth requisite          pam_authtok_get.so.1
gdm auth required           pam_dhkeys.so.1
gdm auth required           pam_unix_cred.so.1
gdm auth required           pam_unix_auth.so.1
gdm auth optional           pam_krb5.so.1
```

The rest of the configuration is as follows:

```
other   account requisite       pam_roles.so.1
other   account required        pam_unix_account.so.1
other   account required        pam_krb5.so.1

other   password required       pam_dhkeys.so.1
other   password requisite      pam_authtok_get.so.1
other   password requisite      pam_authtok_check.so.1
other   password required       pam_authtok_store.so.1
other   password optional       pam_krb5.so.1
```

Non-Kerberos users should specify their respective repositories by using the `-r` option when changing their password with the `passwd(1)` command.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

See Also

`kdestroy(1)`, `kinit(1)`, `kpasswd(1)`, `passwd(1)`, `ktkt_warnd(1M)`, `libpam(3LIB)`, `pam(3PAM)`, `pam_sm(3PAM)`, `pam_sm_acct_mgmt(3PAM)`, `pam_sm_authenticate(3PAM)`, `pam_sm_chauthtok(3PAM)`, `pam_sm_close_session(3PAM)`, `pam_sm_open_session(3PAM)`, `pam_sm_setcred(3PAM)`, `syslog(3C)`, `pam.conf(4)`, `attributes(5)`, `kerberos(5)`, `krb5envvar(5)`

Notes

The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

On successful acquisition of initial credentials (ticket-granting ticket), `ktkt_warnd(1M)` will be notified, to alert the user when the initial credentials are about to expire.

Name pam_krb5_migrate – authentication PAM module for the KerberosV5 auto-migration of users feature

Synopsis `/usr/lib/security/pam_krb5_migrate.so.1`

Description The KerberosV5 auto-migrate service module for PAM provides functionality for the PAM authentication component. The service module helps in the automatic migration of `PAM_USER` to the client's local Kerberos realm, using `PAM_AUTHTOK` (the PAM authentication token associated with `PAM_USER`) as the new Kerberos principal's password.

KerberosV5 Auto-migrate Authentication Module

The KerberosV5 auto-migrate authentication component provides the `pam_sm_authenticate(3PAM)` function to migrate a user who does not have a corresponding `krb5` principal account to the default Kerberos realm of the client.

`pam_sm_authenticate(3PAM)` uses a host-based client service principal, present in the local `keytab` (`/etc/krb5/krb5.keytab`) to authenticate to `kadmind(1M)` (defaults to the `host/nodename.fqdn` service principal), for the principal creation operation. Also, for successful creation of the `krb5` user principal account, the host-based client service principal being used needs to be assigned the appropriate privilege on the master KDC's `kadm5.acl(4)` file. `kadmind(1M)` checks for the appropriate privilege and validates the user password using PAM by calling `pam_authenticate(3PAM)` and `pam_acct_mgmt(3PAM)` for the `k5migrate` service.

If migration of the user to the KerberosV5 infrastructure is successful, the module will inform users about it by means of a `PAM_TEXT_INFO` message, unless instructed otherwise by the presence of the `quiet` option.

The authentication component always returns `PAM_IGNORE` and is meant to be stacked in `pam.conf` with a requirement that it be listed below pam_authtok_get(5) in the authentication stack. Also, if `pam_krb5_migrate` is used in the authentication stack of a particular service, it is mandatory that `pam_krb5(5)` be listed in the PAM account stack of that service for proper operation (see EXAMPLES).

Options The following options can be passed to the KerberosV5 auto-migrate authentication module:

`debug`	Provides `syslog(3C)` debugging information at `LOG_DEBUG` level.
`client_service=`<*service name*>	Name of the service used to authenticate to `kadmind(1M)` defaults to `host`. This means that the module uses `host/`<*nodename.fqdn*> as its client service principal name, KerberosV5 user principal creation operation or <*service*>/<*nodename.fqdn*> if this option is provided.
`quiet`	Do not explain KerberosV5 migration to the user.

This has the same effect as passing the PAM_SILENT flag to pam_sm_authenticate(3PAM) and is useful where applications cannot handle PAM_TEXT_INFO messages.

If not set, the authentication component will issue a PAM_TEXT_INFO message after creation of the Kerberos V5 principal, indicating that it has done so.

expire_pw — Causes the creation of KerberosV5 user principals with password expiration set to now (current time).

Examples

EXAMPLE 1 Sample Entries from pam.conf

The following entries from pam.conf(4) demonstrate the use of the pam_krb5_migrate.so.1 module:

```
login       auth requisite          pam_authtok_get.so.1
login       auth required           pam_dhkeys.so.1
login       auth required           pam_unix_cred.so.1
login       auth sufficient         pam_krb5.so.1
login       auth requisite          pam_unix_auth.so.1
login       auth optional           pam_krb5_migrate.so.1 expire_pw
login       auth required           pam_dial_auth.so.1

other   account requisite       pam_roles.so.1
other   account required        pam_krb5.so.1
other   account required        pam_unix_account.so.1
```

The pam_krb5_migrate module can generally be present on the authentication stack of any service where the application calls pam_sm_authenticate(3PAM) and an authentication token (in the preceding example, the authentication token would be the user's Unix password) is available for use as a Kerberos V5 password.

EXAMPLE 2 Sample Entries from kadm5.acl

The following entries from kadm5.acl(4) permit or deny privileges to the host client service principal:

```
host/*@ACME.COM U root
host/*@ACME.COM ui *
```

The preceding entries permit the pam_krb5_migrate add privilege to the host client service principal of any machine in the ACME.COM KerberosV5 realm, but denies the add privilege to all host service principals for addition of the root user account.

EXAMPLE 3 Sample Entries in `pam.conf` of the Master KDC

The entries below enable `kadmind(1M)` on the master KDC to use the `k5migrate` PAM service in order to validate Unix user passwords for accounts that require migration to the Kerberos realm.

```
k5migrate       auth    required        pam_unix_auth.so.1
k5migrate       account required        pam_unix_account.so.1
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also

`kadmind(1M)`, `syslog(3C)`, `pam_authenticate(3PAM)`, `pam_acct_mgmt(3PAM)`, `pam_sm_authenticate(3PAM)`, `kadm5.acl(4)`, `pam.conf(4)`, `attributes(5)`, `pam_authtok_get(5)`, `pam_krb5(5)`

Name pam_ldap – authentication and account management PAM module for LDAP

Synopsis `/usr/lib/security/pam_ldap.so.1`

Description The `pam_ldap` module implements `pam_sm_authenticate(3PAM)` and `pam_sm_acct_mgmt(3PAM)`, the functions that provide functionality for the PAM authentication and account management stacks. The `pam_ldap` module ties the authentication and account management functionality to the functionality of the supporting LDAP server. For authentication, `pam_ldap` can authenticate the user directly to any LDAP directory server by using any supported authentication mechanism, such as `DIGEST-MD5`. However, the account management component of `pam_ldap` will work only with the Sun Java System Directory Server. The server's user account management must be properly configured before it can be used by `pam_ldap`. Refer to the *Sun Java System Directory Server Administration Guide* for information on how to configure user account management, including password and account lockout policy.

`pam_ldap` must be used in conjunction with the modules that support the UNIX authentication, password, and account management, which are `pam_authtok_get(5)`, `pam_passwd_auth(5)`, `pam_unix_account(5)`, and `pam_unix_auth(5)`. `pam_ldap` is designed to be stacked directly below these modules. If other modules are designed to be stacked in this manner, the modules can be stacked below the `pam_ldap` module. The Examples section shows how the UNIX modules are stacked with `pam_ldap`. When stacked together, the UNIX modules are used to control local accounts, such as `root`. `pam_ldap` is used to control network accounts, that is, LDAP users. For the stacks to work, `pam_unix_auth`, `pam_unix_account`, and `pam_passwd_auth` must be configured with the `binding` control flag and the `server_policy` option. This configuration allows local account override of a network account.

LDAP Authentication Module The LDAP authentication module verifies the identity of a user. The `pam_sm_authenticate(3PAM)` function uses the password entered by the user to attempt to authenticate to the LDAP server. If successful, the user is authenticated. See NOTES for information on password prompting.

The authentication method used is either defined in the client profile , or the authentication method is configured by using the `ldapclient(1M)` command. To determine the authentication method to use, this module first attempts to use the authentication method that is defined, for service `pam_ldap`, for example, `serviceAuthenticationMethod:pam_ldap:sasl/DIGEST-MD5`. If no authentication method is defined, `pam_ldap` uses the default authentication method. If neither are set, the authentication fails. This module skips the configured authentication method if the authentication method is set to `none`.

The following options can be passed to the LDAP service module:

`debug` `syslog(3C)` debugging information at `LOG_DEBUG` level.

`nowarn` Turn off warning messages.

These options are case sensitive and must be used exactly as presented here.

LDAP Account Management Module

The LDAP account management module validates the user's account. The `pam_sm_acct_mgmt(3PAM)` function authenticates to the LDAP server to verify that the user's password has not expired, or that the user's account has not been locked. In the event that there is no user authentication token (`PAM_AUTHTOK`) available, the `pam_sm_acct_mgmt(3PAM)` function attempts to retrieve the user's account status without authenticating to the LDAP server as the user logging in. This procedure will succeed only if the LDAP server is Sun Java System Directory server 5.2 patch 4 or newer. The following options can be passed to the LDAP service module:

`debug` `syslog(3C)` debugging information at `LOG_DEBUG` level.

`nowarn` Turn off warning messages.

These options are case sensitive, and the options must be used exactly as presented here.

LDAP Password Management Module

LDAP password management is no longer supported by `pam_ldap`. Use `pam_authtok_store(5)` instead of `pam_ldap` for password change. `pam_authtok_store(5)` handles both the local and LDAP accounts and updates the passwords in all the repositories configured by `nsswitch.conf(4)`.

Errors

The authentication service returns the following error codes:

`PAM_SUCCESS`	The uthentication was successful.
`PAM_MAXTRIES`	The maximum number of authentication attempts was exceeded.
`PAM_AUTH_ERR`	The authentication failed.
`PAM_USER_UNKNOWN`	No account is present for the user.
`PAM_BUF_ERR`	A memory buffer error occurred.
`PAM_SYSTEM_ERR`	A system error occurred.
`PAM_IGNORE`	The user's account was inactivated.

The account management service returns the following error codes:

`PAM_SUCCESS`	The user was allowed access to the account.
`PAM_NEW_AUTHTOK_REQD`	A new authentication token is required.
`PAM_ACCT_EXPIRED`	The user account has expired.
`PAM_PERM_DENIED`	The user was denied access to the account at this time.
`PAM_USER_UNKNOWN`	No account is present for the user.
`PAM_BUF_ERROR`	A memory buffer error occurred.
`PAM_SYSTEM_ERR`	A system error occurred.

Examples

EXAMPLE 1 Using `pam_ldap` With Authentication

The following is a configuration for the login service when using `pam_ldap`. The service name `login` can be substituted for any other authentication service such as `dtlogin` or `su`. Lines that begin with the # symbol are comments and are ignored.

```
# Authentication management for login service is stacked.
# If pam_unix_auth succeeds, pam_ldap is not invoked.
# The control flag "binding" provides a local overriding
# remote (LDAP) control. The "server_policy" option is used
# to tell pam_unix_auth.so.1 to ignore the LDAP users.

login   auth requisite  pam_authtok_get.so.1
login   auth required   pam_dhkeys.so.1
login   auth required   pam_unix_cred.so.1
login   auth binding    pam_unix_auth.so.1 server_policy
login   auth required   pam_ldap.so.1
```

EXAMPLE 2 Using `pam_ldap` With Account Management

The following is a configuration for account management when using `pam_ldap`. Lines that begin with the # symbol are comments and are ignored.

```
# Account management for all services is stacked
# If pam_unix_account succeeds, pam_ldap is not invoked.
# The control flag "binding" provides a local overriding
# remote (LDAP) control. The "server_policy" option is used
# to tell pam_unix_account.so.1 to ignore the LDAP users.

other   account  requisite      pam_roles.so.1
other   account  binding        pam_unix_account.so.1 server_policy
other   account  required       pam_ldap.so.1
```

EXAMPLE 3 Using `pam_authtok_store` With Password Management For Both Local and LDAP Accounts

The following is a configuration for password management when using `pam_authtok_store`. Lines that begin with the # symbol are comments and are ignored.

```
# Password management (authentication)
# The control flag "binding" provides a local overriding
# remote (LDAP) control. The server_policy option is used
# to tell pam_passwd_auth.so.1 to ignore the LDAP users.

passwd  auth binding  pam_passwd_auth.so.1 server_policy
passwd  auth required pam_ldap.so.1

# Password management (updates)
```

EXAMPLE 3 Using `pam_authtok_store` With Password Management For Both Local and LDAP Accounts *(Continued)*

```
# This updates passwords stored both in the local /etc
# files and in the LDAP directory. The "server_policy"
# option is used to tell pam_authtok_store to
# follow the LDAP server's policy when updating
# passwords stored in the LDAP directory

other password required   pam_dhkeys.so.1
other password requisite  pam_authtok_get.so.1
other password requisite  pam_authtok_check.so.1
other password required   pam_authtok_store.so.1 server_policy
```

Files

`/var/ldap/ldap_client_file`
`/var/ldap/ldap_client_cred` — The LDAP configuration files of the client. Do not manually modify these files, as these files might not be human readable. Use `ldapclient(1M)` to update these files.

`/etc/pam.conf` — PAM configuration file.

Attributes

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

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

See Also

`ldap(1)`, `idsconfig(1M)`, `ldap_cachemgr(1M)`, `ldapclient(1M)`, `libpam(3LIB)`, `pam(3PAM)`, `pam_sm_acct_mgmt(3PAM)`, `pam_sm_authenticate(3PAM)`, `pam_sm_chauthtok(3PAM)`, `pam_sm_close_session(3PAM)`, `pam_sm_open_session(3PAM)`, `pam_sm_setcred(3PAM)`, `syslog(3C)`, `pam.conf(4)`, `attributes(5)`, `pam_authtok_check(5)`, `pam_authtok_get(5)`, `pam_authtok_store(5)`, `pam_passwd_auth(5)`, `pam_unix_account(5)`, `pam_unix_auth(5)`

Notes

The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own `PAM` handle.

The previously supported `use_first_pass` and `try_first_pass` options are obsolete in this version, are no longer needed, can safely be removed from `pam.conf(4)`, and are silently ignored. They might be removed in a future release. Password prompting must be provided for by stacking `pam_authtok_get(5)` before `pam_ldap` in the `auth` and `password` module stacks and `pam_passwd_auth(5)` in the `passwd` service `auth` stack (as described in the EXAMPLES section). The previously supported password update function is replaced in this release by the previously recommended use of `pam_authtok_store` with the `server_policy` option (as described in the EXAMPLES section).

The functions: pam_sm_setcred(3PAM), pam_sm_chauthtok(3PAM), pam_sm_open_session(3PAM), and pam_sm_close_session(3PAM) do nothing and return PAM_IGNORE in pam_ldap.

Name pam_list – PAM account management module for UNIX

Synopsis

```
pam_list.so.1
```

Description The `pam_list` module implements `pam_sm_acct_mgmt(3PAM)`, which provides functionality to the PAM account management stack. The module provides functions to validate that the user's account is valid on this host based on a list of users and/or netgroups in the given file. The users and netgroups are separated by newline character. Netgroups are specified with character '@' as prefix before name of netgroup in the list. The maximum line lenght is 1023 characters.

The username is the value of `PAM_USER`. The host is the value of `PAM_RHOST` or, if `PAM_RHOST` is not set, the value of the localhost as returned by `gethostname(3C)` is used.

If neither of the `allow`, `deny`, or `compat` options are specified, the module will look for +/- entries in the local `/etc/passwd` file. If this style is used, `nsswitch.conf(4)` must not be configured with `compat` for the `passwd` database. If no relevant +/- entry exists for the user, `pam_list` is not participating in result.

If `compat` option is specified then the module will look for +/- entries in the local `/etc/passwd` file. Other entries in this file will be counted as + entries. If no relevant entry exits for the user, `pam_list` will deny the access.

The following options can be passed to the module:

`allow=`	The full pathname to a file of allowed users and/or netgroups. Only one of `allow=` or `deny=` can be specified.
`compat`	Activate `compat` mode.
`deny=`	The full pathname to a file of denied users and/or netgroups. Only one of `deny=` or `allow=` can be specified.
`debug`	Provide `syslog(3C)` debugging information at the `LOG_AUTH` | `LOG_DEBUG` level.
`user`	The module should only perform netgroup matches on the username. This is the default option.
`nouser`	The username should not be used in the netgroup match.
`host`	Only the host should be used in netgroup matches.
`nohost`	The hostname should not be used in netgroup matches.
`user_host_exact`	The user and hostname must be in the same netgroup.

Errors The following error values are returned:

`PAM_SERVICE_ERR`	An invalid set of module options was given in the `pam.conf(4)` for this module, or the `user/netgroup` file could not be opened.

PAM_BUF_ERR	A memory buffer error occurred.
PAM_IGNORE	The module is ignored, as it is not participating in the result.
PAM_PERM_DENIED	The user is not on the allow list or is on the deny list.
PAM_SUCCESS	The account is valid for use at this time.
PAM_USER_UNKNOWN	No account is present for the user

Examples

EXAMPLE 1 Using pam_list in default mode

/etc/pam.conf modification looks like:

```
other   account requisite       pam_roles.so.1
other   account required        pam_unix_account.so.1
other   account required        pam_list.so.1
```

In the case of default mode or compat mode, the important lines in /etc/passwd appear as follows:

```
+loginname    - user is approved
-loginname    - user is disapproved
+@netgroup    - netgroup members are approved
-@netgroup    - netgroup members are disapproved
```

EXAMPLE 2 Using pam_list with allow file

/etc/pam.conf modification looks like:

```
other   account requisite       pam_roles.so.1
other   account required        pam_unix_account.so.1
other   account required        pam_list.so.1 allow=/etc/user.allow
```

/etc/users.allow contains:

```
root
localloginname
remoteloginname
@netgroup
```

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with exceptions

The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

See Also pam(3PAM), pam_authenticate(3PAM), pam_sm_acct_mgmt(3PAM), syslog(3C), libpam(3LIB), nsswitch.conf(4), pam.conf(4), attributes(5)

Name pam_passwd_auth – authentication module for password

Synopsis

```
pam_passwd_auth.so.1
```

Description `pam_passwd_auth` provides authentication functionality to the password service as implemented by `passwd(1)`. It differs from the standard PAM authentication modules in its prompting behavior. It should be the first module on the password service authentication stack.

The name of the user whose password attributes are to be updated must be present in the `PAM_USER` item. This can be accomplished due to a previous call to `pam_start(3PAM)`, or explicitly set by `pam_set_item(3PAM)`. Based on the current user-id and the repository that is to by updated, the module determines whether a password is necessary for a successful update of the password repository, and if so, which password is required.

The following options can be passed to the module:

`debug`	`syslog(3C)` debugging information at the `LOG_DEBUG` level
`nowarn`	Turn off warning messages
`server_policy`	If the account authority for the user, as specified by `PAM_USER`, is a server, do not apply the Unix policy from the passwd entry in the name service switch.

Errors The following error codes are returned:

`PAM_BUF_ERR`	Memory buffer error
`PAM_IGNORE`	Ignore module, not participating in result
`PAM_SUCCESS`	Successfully obtains authentication token
`PAM_SYSTEM_ERR`	System error

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

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

See Also `passwd(1)`, `pam(3PAM)`, `pam_authenticate(3PAM)`, `pam_start(3PAM)`, `pam_set_item(3PAM)`, `syslog(3C)`, `libpam(3LIB)`, `pam.conf(4)`, `attributes(5)`, `pam_authtok_check(5)`, `pam_authtok_get(5)`, `pam_authtok_store(5)`, `pam_dhkeys(5)`, `pam_unix_account(5)`, `pam_unix_auth(5)`, `pam_unix_session(5)`

Notes The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

This module relies on the value of the current real UID, this module is only safe for MT-applications that don't change UIDs during the call to pam_authenticate(3PAM).

The pam_unix(5) module is no longer supported. 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_pkcs11 – PAM Authentication Module for the PKCS#11 token libraries

Synopsis `pam_pkcs11.so [debug] [config_file=`*filename*`]`

Description The `pam_pkcs11` module implements `pam_sm_authenticate(3PAM)`, which provides functionality to the PAM authentication stack. This module allows a user to login a system, using a X.509 certificate and its dedicated private key stored in a PKCS#11 token. This module currently supports the RSA algorithm only.

To verify the dedicated private key is truly associated with the X.509 certificate, the following verification procedure is performed in this module by default:

- Generate 128 random byte data
- Sign the random data with the private key and get a signature. This step is done in the PKCS#11 token.
- Verify the signature using the public key extracted from the certificate.

For the verification of the users' certificates, locally stored CA certificates as well as either online or locally accessible CRLs are used.

PAM CONFIGURATION The `pam_pkcs11.so` service module can be used in the `<auth>` PAM chain. The program that needs a PAM service should be configured in the `/etc/pam.conf` file. For details on how to configure PAM services, see `pam.conf(4)`.

The following example uses only `pam_pkcs11` for authentication:

```
login auth requisite pam_pkcs11.so.1
login autho required pam_unix_cred.so.1
```

The following example uses `pam_pkcs11` for authentication with fallback to standard UNIX authentication:

```
login auth sufficient pam_pkcs11.so.1
login auth requisite  pam_authtok_get.so.1
login auth required   pam_dhkeys.so.1
login auth required   pam_unix_cred.so.1
login auth required   pam_unix_auth.so.1
```

PAM_PKCS11 CONFIGURATION To configure the `pam_pkcs11` module, you must have the following information:

- Which PKCS #11 token you are going to use
- Which mapper(s) you need, and if needed, how to create and edit the related mapping files
- The root Certificate Authority files, and if required, the Certificate Revocation Lists files
- The list of authorized users to login, and their corresponding certificates

To configure the `pam_pkcs11` module, you need to modify the `pam_pkcs11.conf` configuration file which is in the `/etc/security/pam_pkcs11` directory by default. For

detailed information on how to configure the pam_pkcs11 module, see the *PAM-PKCS11 User Manual*, available at the http://www.opensc-project.org/ web site, under the PAM PKCS#11 link.

The following example illustrates how to configure the pam_pkcs11 module for a user whose certificate and private key are stored in the Solaris pkcs11_softtoken keystore. This example uses the default certificate verification policy.

- Set up the PKCS#11 module.

  On Solaris, the PKCS#11 module should be set to /usr/lib/libpkcs11.so.1, the PKCS#11 Cryptographic Framework library.

- Set up the slot_description entry.

  Specifies the slot to be used. For example, slot_description = "Sun Crypto Softtoken". The default value for this entry is none which means to use the first slot with an available token.

  An administrator can use the cryptoadm list -vcommand to find all the available slots and their slot descriptions. For more information, see libpkcs11(3LIB) and cryptoadm(1M).

- Install or create user certificates and its dedicated private keys in the specific PKCS#11 token.

- Set up the certificate verification policy (cert_policy). If needed, set up CA certificate and CRL files.

  The certificate verification policy includes:

none	Perform no verification
ca	Perform CA check
signature	Perform a signature check to ensure that private and public key matches
crl_*xxx*	Perform various certificate revocation checking

  As this example uses the default policy, cert_policy = ca,signature, an administer needs to set up the CA certificates.

  – Copy the CA certificate to the /etc/security/pam_pkcs11/cacerts directory.

    A certificate that is self-signed is its own CA certificate. Therefore, in this example, the certificate is placed both in the Softtoken keystore and in the CA certificate directory.

  – Make hash links for CA certificates

  ```
  $ /etc/security/pam_pkcs11/make_hash_link.sh \
        /etc/security/pam_pkcs11/cacerts
  ```

- Set up the mappers and mapfiles.

When a X509 certificate is provided, there are no direct ways to map a certificate to a login. The `pam_pkcs11` module provides a configurable way with mappers to specify `cert-to-user` mapping.

Many mappers are provided by the `pam_pkcs11` module, for example, the common name (CN) mapper, the digest mapper, the Email mapper, or the LDAP mapper.

A user can configure a mapper list in the `pam_pkcs11.conf` file. The mappers in the list are used sequentially until the certificate is successfully matched with the user.

The default mapper list is as follows:

```
use_mappers = digest, cn, pwent, uid, mail, subject, null;
```

Some mappers do not require the specification of a mapfile, for example, the common name mapper. Other mappers require mapfiles, for example, the digest mapper. Some sample mapping files can be found in the `/etc/security/pam_pkcs11` directory.

Options The following options are supported:

`config_file=`*filename* Specify the configuration file. The default value is `/etc/security/pam_pkcs11/pam_pkcs11.conf`.

`debug` Enable debugging output.

Files `/usr/lib/security/pam_pkcs11.so`
`pam_pkcs11` module

`/usr/lib/pam_pkcs11/ldap_mapper.so`
Mapper module.

`/usr/lib/pam_pkcs11/opensc_mapper.so`
Mapper module.

`/usr/lib/pam_pkcs11/openssh_mapper.so`
Mapper module.

`/etc/security/pam_pkcs11/pam_pkcs11.conf`
Configuration file.

`/etc/security/pam_pkcs11/cacerts`
Configuration directory. Stores the CA certificates.

`/etc/security/pam_pkcs11/crls`
Configuration directory. Stores the CRL files.

`/etc/security/pam_pkcs11/digest_mapping.example`
Sample mapfile.

`/etc/security/pam_pkcs11/subject_mapping.example`
Sample mapfile.

`/etc/security/pam_pkcs11/mail_mapping.example`
Sample mapfile.

`/etc/security/pam_pkcs11/make_hash_link.sh`
Sample script.

Authors `PAM-pkcs11` was originally written by MarioStrasser , `mast@gmx.net`.

Newer versions are from Juan Antonio Martinez, `jonsito@teleline.es`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWpampkcs11u, SUNWpampkcs11r, SUNWpampkcs11-docs
Interface Stability	Uncommitted

See Also `pkcs11_inspect(1)`, `pklogin_finder(1)`, `cryptoadm(1M)`, `libpkcs11(3LIB)libpkcs11(3LIB)pam_sm_authenticate(3PAM)`, `pam.conf(4)`, `attributes(5)`, `pkcs11_softtoken(5)`

PAM-PKCS11 User Manual, available at the `http://www.opensc-project.org/` web site, under the `PAM PKCS#11` link.

Name pam_rhosts_auth – authentication management PAM module using ruserok()

Synopsis `/usr/lib/security/pam_rhosts_auth.so.1`

Description The rhosts PAM module, `/usr/lib/security/pam_rhosts_auth.so.1`, authenticates a user via the `rlogin` authentication protocol. Only `pam_sm_authenticate()` is implemented within this module. `pam_sm_authenticate()` uses the `ruserok(3SOCKET)` library function to authenticate the `rlogin` or `rsh` user. `pam_sm_setcred()` is a null function.

`/usr/lib/security/pam_rhosts_auth.so.1` is designed to be stacked on top of the `/usr/lib/security/pam_unix.so.1` module for both the `rlogin` and `rsh` services. This module is normally configured as *sufficient* so that subsequent authentication is performed only on failure of `pam_sm_authenticate()`. The following option may be passed in to this service module:

`debug` `syslog(3C)` debugging information at LOG_DEBUG level.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

See Also `pam(3PAM)`, `pam_authenticate(3PAM)`, `ruserok(3SOCKET)`, `syslog(3C)`, `libpam(3LIB)`, `pam.conf(4)`, `attributes(5)`

Notes The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

Name pam_roles – Solaris Roles account management module

Synopsis `pam_roles.so.1`

Description The `pam_roles` module implements `pam_sm_acct_mgmt(3PAM)`. It provides functionality to verify that a user is authorized to assume a role. It also prevents direct logins to a role. The `user_attr(4)` database is used to determine which users can assume which roles.

The PAM items `PAM_USER` and `PAM_AUSER`, and `PAM_RHOST` are used to determine the outcome of this module. `PAM_USER` represents the new identity being verified. `PAM_AUSER`, if set, represents the user asserting a new identity. If `PAM_AUSER` is not set, the real user ID of the calling service implies that the user is asserting a new identity. Notice that root can never have roles.

This module is generally stacked above the `pam_unix_account(5)` module.

The following options are interpreted:

`allow_remote` Allows a remote service to specify the user to enter as a role.

`debug` Provides `syslog(3C)` debugging information at the `LOG_DEBUG` level.

Errors The following values are returned:

`PAM_IGNORE` If the type of the new user identity (`PAM_USER`) is "`normal`". Or, if the type of the new user identity is "`role`" and the user asserting the new identity (`PAM_AUSER`) has the new identity name in its list of roles.

`PAM_USER_UNKNOWN` No account is present for user.

`PAM_PERM_DENIED` If the type of the new user identity (`PAM_USER`) is "`role`" and the user asserting the new identity (`PAM_AUSER`) does not have the new identity name in its list of roles.

Examples **EXAMPLE 1** Using the `pam_roles.so.1` Module

The following are sample entries from `pam.conf(4)`. These entries demonstrate the use of the `pam_roles.so.1` module:

```
cron account required pam_unix_account.so.1
#
other account requisite pam_roles.so.1
other account required pam_unix_account.so.1
#
```

The `cron` service does not invoke `pam_roles.so.1`. Delayed jobs are independent of role assumption. All other services verify that roles cannot directly login. The "`su`" service (covered by the "`other`" service entry) verifies that if the new user is a role, the calling user is authorized for that role.

EXAMPLE 2 Allowing Remote Roles

Remote roles should only be allowed from remote services that can be trusted to provide an accurate PAM_AUSERname. This trust is a function of the protocol (such as sshd-hostbased).

The following is a sample entry for a pam.conf(4) file. It demonstrates the use of pam_roles configuration for remote roles for the sshd-hostbased service.

```
sshd-hostbased account requisite pam_roles.so.1 allow_remote
sshd-hostbased account required pam_unix_account
```

Attributes

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

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

See Also

roles(1), sshd(1M), su(1M), libpam(3LIB), pam(3PAM), pam_acct_mgmt(3PAM), pam_setcred(3PAM), pam_set_item(3PAM), pam_sm_acct_mgmt(3PAM), syslog(3C), pam.conf(4), user_attr(4), attributes(5), 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), pam_unix_session(5)

Notes

The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

This module should never be stacked alone. It never returns PAM_SUCCESS, as it never makes a positive decision.

The allow_remote option should only be specified for services that are trusted to correctly identify the remote user (that is, sshd-hostbased).

PAM_AUSER has replaced PAM_RUSER whose definition is limited to the rlogin/rsh untrusted remote user name. See pam_set_item(3PAM).

Name pam_sample – a sample PAM module

Synopsis `/usr/lib/security/pam_sample.so.1`

Description The SAMPLE service module for PAM is divided into four components: authentication, account management, password management, and session management. The sample module is a shared object that is dynamically loaded to provide the necessary functionality.

Sample Authentication Component The SAMPLE authentication module provides functions to test the PAM framework functionality using the `pam_sm_authenticate(3PAM)` call. The SAMPLE module implementation of the `pam_sm_authenticate(3PAM)` function compares the user entered password with the password set in the `pam.conf(4)` file, or the string `test` if a default test password has not been set. The following options can be passed in to the SAMPLE Authentication module:

`debug`	Syslog debugging information at the `LOG_DEBUG` level.
`pass=newone`	Sets the password to be `newone`.
`first_pass_good`	The first password is always good when used with the use_first_pass or try_first_pass option.
`first_pass_bad`	The first password is always bad when used with the use_first_pass or try_first_pass option.
`always_fail`	Always returns `PAM_AUTH_ERR`.
`always_succeed`	Always returns `PAM_SUCCESS`.
`always_ignore`	Always returns `PAM_IGNORE`.
`use_first_pass`	Use the user's initial password (entered when the user is authenticated to the first authentication module in the stack) to authenticate with the SAMPLE module. If the passwords do not match, or if this is the first authentication module in the stack, quit and do not prompt the user for a password. It is recommended that this option only be used if the SAMPLE authentication module is designated as *optional* in the `pam.conf` configuration file.
`try_first_pass`	Use the user's initial password (entered when the user is authenticated to the first authentication module in the stack) to authenticate with the SAMPLE module. If the passwords do not match, or if this is the first authentication module in the stack, prompt the user for a password. The SAMPLE module `pam_sm_setcred(3PAM)` function always returns `PAM_SUCCESS`.

Sample Account Management Component The SAMPLE Account Management Component implements a simple access control scheme that limits machine access to a list of authorized users. The list of authorized users is supplied

as option arguments to the entry for the SAMPLE account management PAM module in the `pam.conf` file. Note that the module always permits access to the root super user.

The option field syntax to limit access is shown below: allow= *name[,name]* allow= *name [allow=name]*

The example `pam.conf` show below permits only larry to `login` directly. `rlogin` is allowed only for don and larry. Once a user is logged in, the user can use `su` if the user are sam or eric.

login	account	require	pam_sample.so.1	allow=larry
gdm	account	require	pam_sample.so.1	allow=larry
rlogin	account	require	pam_sample.so.1	allow=don allow=larry
su	account	require	pam_sample.so.1	allow=sam,eric

The debug and nowarn options are also supported.

Sample Password Management Component

The SAMPLE Password Management Component function (`pam_sm_chauthtok(3PAM)`), always returns `PAM_SUCCESS`.

Sample Session Management Component

The SAMPLE Session Management Component functions (`pam_sm_open_session(3PAM)`, `pam_sm_close_session(3PAM)`) always return `PAM_SUCCESS`.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

See Also

`pam(3PAM)`, `pam_sm_authenticate(3PAM)`, `pam_sm_chauthtok(3PAM)`, `pam_sm_close_session(3PAM)`, `pam_sm_open_session(3PAM)`, `pam_sm_setcred(3PAM)`, `libpam(3LIB)`, `pam.conf(4)`, `attributes(5)`

Warnings

This module should never be used outside of a closed debug environment. The examples of the `use_first_pass` and `try_first_pass` options are obsolete for all other Solaris delivered PAM service modules

Notes

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

Name pam_smbfs_login – PAM user credential authentication module for SMB/CIFS client login

Synopsis `pam_smb_cred.so.1`

Description The `pam_smbfs_login` module implements `pam_sm_setcred(3PAM)` to provide functions that act equivalently to the `smbutil(1)` login command.

This optional functionality is meant to be used only in environments that do not run Active Directory or Kerberos, but which synchronize passwords between Solaris clients and their CIFS/SMB servers.

This module permits the login password to be stored as if the `smbutil(1)` login command was used to store a password for PAM_USER in the user or system default domain. The choice of default domain is the first of the following:

–Domain entry specified in the default section of the `$HOME/.nsmbrc` file, if readable.
–Domain entry specified in the default section shown by the sharectl get smbfs command.
–String WORKGROUP.

Because `pam_smbfs_login` runs as root during the login process, a `$HOME/.nsmbrc` file accessed through NFS may only be readable if the file permits reads by others. This conflicts with the requirement that passwords stored in `$HOME/.nsmbrc` are ignored when permissions are open.

To use this functionality, add the following line to the `/etc/pam.conf` file:

```
login  auth optional    pam_smbfs_login.so.1
```

Authentication service modules must implement both `pam_sm_authenticate(3PAM)` and `pam_sm_setcred(3PAM)`. In this module, `pam_sm_authenticate(3PAM)` always returns `PAM_IGNORE`.

The `pam_sm_setcred(3PAM)` function accepts the following flags:

`PAM_REFRESH_CRED`
Returns PAM_IGNORE.

`PAM_SILENT`
Suppresses messages.

`PAM_ESTABLISH_CRED`
`PAM_REINITIALIZE_CRED`
Stores the authentication token for PAM_USER in the same manner as the `smbutil(1)` login command.

`PAM_DELETE_CRED`
Deletes the stored password for PAM_USER in the same manner as the `smbutil(1)` logout command.

The following options can be passed to the `pam_smbfs_login` module:

`debug`
Produces `syslog(3C)` debugging information at the LOG_AUTH or LOG_DEBUG level.

`nowarn`
Suppresses warning messages.

Files `$HOME/.nsmbrc` Find default domain, if present.

Errors Upon successful completion of `pam_sm_setcred(3PAM)`, `PAM_SUCCESS` is returned. The following error codes are returned upon error:

`PAM_USER_UNKNOWN`
User is unknown.

`PAM_AUTHTOK_ERR`
Password is bad.

`PAM_AUTH_ERR`
Domain is bad.

`PAM_SYSTEM_ERR`
System error.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT Level	MT-Safe with exceptions

See Also `smbutil(1)`, `syslog(3C)`, `libpam(3LIB)`, `pam(3PAM)`, `pam_setcred(3PAM)`, `pam_sm(3PAM)`, `pam_sm_authenticate(3PAM)`, `pam_sm_chauthtok(3PAM)`, `pam_sm_setcred(3PAM)`, `pam.conf(4)`, `attributes(5)`, `smbfs(7FS)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

Name pam_smb_passwd – SMB password management module

Synopsis `pam_smb_passwd.so.1`

Description The `pam_smb_passwd` module enhances the PAM password management stack. This functionality supports the changing or adding of SMB passwords for local Solaris users. The Solaris CIFS server uses SMB passwords to authenticate connected Solaris users. This module includes the `pam_sm_chauthtok(3PAM)` function.

The `pam_sm_chauthtok()` function accepts the following flags:

`PAM_PRELIM_CHECK`
Always returns `PAM_IGNORE`.

`PAM_SILENT`
Suppresses messages.

`PAM_UPDATE_AUTHTOK`
Updates or creates a new CIFS local LM/NTLM hash for the user that is specified in `PAM_USER` by using the authentication information found in `PAM_AUTHTOK`. The LM hash is only created if the `smbd/lmauth_level` property value of the `smb/server` service is set to 3 or less. `PAM_IGNORE` is returned if the user is not in the local `/etc/passwd` repository.

The following options can be passed to the `pam_smb_passwd` module:

`debug`
Produces `syslog(3C)` debugging information at the `LOG_AUTH` or `LOG_DEBUG` level.

`nowarn`
Suppresses warning messages.

Files `/var/smb/smbpasswd`
Stores SMB passwords for Solaris users.

Errors Upon successful completion of `pam_sm_chauthtok()`, `PAM_SUCCESS` is returned. The following error codes are returned upon error:

`PAM_AUTHTOK_ERR`
Authentication token manipulation error

`PAM_AUTHTOK_LOCK_BUSY`
SMB password file is locked

`PAM_PERM_DENIED`
Permissions are insufficient for accessing the SMB password file

`PAM_SYSTEM_ERR`
System error

`PAM_USER_UNKNOWN`
User is unknown

Attributes See the attributes(5) man page for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT Level	MT-Safe with exceptions

See Also `smbd`(1M), `syslog`(3C), `libpam`(3LIB), `pam`(3PAM), `pam_chauthtok`(3PAM), `pam_sm`(3PAM), `pam_sm_chauthtok`(3PAM), `pam.conf`(4), attributes(5)

Notes The interfaces in `libpam`(3LIB) are MT-Safe *only* if each thread within the multi-threaded application uses its own PAM handle.

The `pam_smb_passwd.so.1` module should be stacked following all password qualification modules in the PAM password stack.

Name pam_tsol_account – PAM account management module for Trusted Extensions

Synopsis `/usr/lib/security/pam_tsol_account.so.1`

Description The Solaris Trusted Extensions service module for PAM, `/usr/lib/security/pam_tsol_account.so.1`, checks account limitations that are related to labels. The `pam_tsol_account.so.1` module is a shared object that can be dynamically loaded to provide the necessary functionality upon demand. Its path is specified in the PAM configuration file.

`pam_tsol_account.so.1` contains a function to perform account management, `pam_sm_acct_mgmt()`. The function checks for the allowed label range for the user. The allowable label range is set by the defaults in the `label_encodings(4)` file. These defaults can be overridden by entries in the `user_attr(4)` database.

By default, this module requires that remote hosts connecting to the global zone must have a CIPSO host type. To disable this policy, add the `allow_unlabeled` keyword as an option to the entry in `pam.conf(4)`, as in:

```
other  account required    pam_tsol_account allow_unlabeled
```

Options The following options can be passed to the module:

`allow_unlabeled`	Allows remote connections from hosts with unlabeled template types.
`debug`	Provides debugging information at the `LOG_DEBUG` level. See `syslog(3C)`.

Return Values The following values are returned:

`PAM_SUCCESS`	The account is valid for use at this time and label.
`PAM_PERM_DENIED`	The current process label is outside the user's label range, or the label information for the process is unavailable, or the remote host type is not valid.
Other values	Returns an error code that is consistent with typical PAM operations. For information on error-related return values, see the `pam(3PAM)` man page.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT Level	MT-Safe with exceptions

The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

See Also keylogin(1), libpam(3LIB), pam(3PAM), pam_sm_acct_mgmt(3PAM), pam_start(3PAM), syslog(3C), label_encodings(4), pam.conf(4), user_attr(4), attributes(5)

Chapter 17, "Using PAM," in *System Administration Guide: Security Services*

Notes The functionality described on this manual page is available only if the system is configured with Trusted Extensions.

Name pam_unix_account – PAM account management module for UNIX

Synopsis `pam_unix_account.so.1`

Description `pam_unix_account` module implements `pam_sm_acct_mgmt()`, which provides functionality to the PAM account management stack. The module provides functions to validate that the user's account is not locked or expired and that the user's password does not need to be changed. The module retrieves account information from the configured databases in `nsswitch.conf(4)`.

The following options can be passed to the module:

`debug`	`syslog(3C)` debugging information at the `LOG_DEBUG` level
`nowarn`	Turn off warning messages
`server_policy`	If the account authority for the user, as specified by `PAM_USER`, is a server, do not apply the Unix policy from the passwd entry in the name service switch.

Errors The following values are returned:

`PAM_UNIX_ACCOUNT`	User account has expired
`PAM_AUTHTOK_EXPIRED`	Password expired and no longer usable
`PAM_BUF_ERR`	Memory buffer error
`PAM_IGNORE`	Ignore module, not participating in result
`PAM_NEW_AUTHTOK_REQD`	Obtain new authentication token from the user
`PAM_PERM_DENIED`	The account is locked or has been inactive for too long
`PAM_SERVICE_ERR`	Error in underlying service module
`PAM_SUCCESS`	The account is valid for use at this time
`PAM_USER_UNKNOWN`	No account is present for the user

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

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

See Also `pam(3PAM)`, `pam_authenticate(3PAM)`, `syslog(3C)`, `libpam(3LIB)`, `pam.conf(4)`, `nsswitch.conf(4)`, `attributes(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

Attempts to validate locked accounts are logged via `syslog(3C)` to the `LOG_AUTH` facility with a `LOG_NOTICE` severity.

Name pam_unix_auth – PAM authentication module for UNIX

Synopsis `pam_unix_auth.so.1`

Description The `pam_unix_auth` module implements `pam_sm_authenticate()`, which provides functionality to the PAM authentication stack. It provides functions that use `crypt(3C)` to verify that the password contained in the PAM item `PAM_AUTHTOK` is the correct password for the user specified in the item `PAM_USER`. If `PAM_REPOSITORY` is specified, then user's password is fetched from that repository. Otherwise, the default `nsswitch.conf(4)` repository is searched for that user.

For accounts in the name services which support automatic account locking, the account may be configured to be automatically locked (see `user_attr(4)` and `policy.conf(4)`) after multiple failed login attempts. For accounts that are configured for automatic locking, if authentication failure is to be returned, the failed login counter is incremented upon each failure. If the number of successive failures equals or exceeds `RETRIES` as defined in `login(1)`, the account is locked and `PAM_MAXTRIES` is returned. Currently, only the "files" repository (see `passwd(4)` and `shadow(4)`) supports automatic account locking. A successful authentication by this module clears the failed login counter and reports the number of failed attempts since the last successful authentication.

Authentication service modules must implement both `pam_sm_authenticate()` and `pam_sm_setcred()`. To allow the authentication portion of UNIX authentication to be replaced, `pam_sm_setcred()` in this module always returns `PAM_IGNORE`. This module should be stacked with `pam_unix_cred(5)` to ensure a successful return from `pam_setcred(3PAM)`.

The following options can be passed to the module:

`nowarn`
Turn off warning messages.

`server_policy`
If the account authority for the user, as specified by `PAM_USER`, is a server, do not apply the UNIX policy from the `passwd` entry in the name service switch.

`nolock`
Regardless of the automatic account locking setting for the account, do not lock the account, increment or clear the failed login count. The `nolock` option allows for exempting account locking on a per service basis.

Errors The following error codes are returned from `pam_sm_authenticate()`:

`PAM_AUTH_ERR`
Authentication failure.

`PAM_BUF_ERR`
Memory buffer error.

`PAM_IGNORE`
Ignores module, not participating in result.

PAM_MAXTRIES
Maximum number of retries exceeded.

PAM_PERM_DENIED
Permission denied.

PAM_SUCCESS
Successfully obtains authentication token.

PAM_SYSTEM_ERR
System error.

PAM_USER_UNKNOWN
No account present for user.

The following error codes are returned from pam_sm_setcred():

PAM_IGNORE
Ignores this module regardless of the control flag.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT Level	MT-Safe with exceptions

See Also

login(1), passwd(1), useradd(1M), usermod(1M), roleadd(1M), rolemod(1M), crypt(3C), libpam(3LIB), pam(3PAM), pam_authenticate(3PAM), pam_setcred(3PAM), syslog(3C), pam.conf(4), passwd(4), policy.conf(4), nsswitch.conf(4), shadow(4), user_attr(4), attributes(5), 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_session(5)

Notes

The interfaces in libpam(3LIB) are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The pam_unix(5) module is no longer supported. 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_setcred(3PAM), pam_unix_account(5), pam_unix_cred(5), pam_unix_session(5).

If the PAM_REPOSITORY *item_type* is set and a service module does not recognize the type, the service module does not process any information, and returns PAM_IGNORE. If the PAM_REPOSITORY *item_type* is not set, a service module performs its default action.

Name pam_unix_cred – PAM user credential authentication module for UNIX

Synopsis `pam_unix_cred.so.1`

Description The `pam_unix_cred` module implements `pam_sm_setcred(3PAM)`. It provides functions that establish user credential information. It is a module separate from the `pam_unix_auth(5)` module to allow replacement of the authentication functionality independently from the credential functionality.

The `pam_unix_cred` module must always be stacked along with whatever authentication module is used to ensure correct credential setting.

Authentication service modules must implement both `pam_sm_authenticate()` and `pam_sm_setcred()`.

`pam_sm_authenticate()` in this module always returns `PAM_IGNORE`.

`pam_sm_setcred()` initializes the user's project, privilege sets and initializes or updates the user's audit context if it hasn't already been initialized. The following flags may be set in the flags field:

`PAM_ESTABLISH_CRED`
`PAM_REFRESH_CRED`
`PAM_REINITIALIZE_CRED`
: Initializes the user's project to the project specified in `PAM_RESOURCE`, or if `PAM_RESOURCE` is not specified, to the user's default project. Establishes the user's privilege sets.

  If the audit context is not already initialized and auditing is configured, these flags cause the context to be initialized to that of the user specified in `PAM_AUSER` (if any) merged with the user specified in `PAM_USER` and host specified in `PAM_RHOST`. If `PAM_RHOST` is not specified, `PAM_TTY` specifies the local terminal name. Attributing audit to `PAM_AUSER` and merging `PAM_USER` is required for correctly attributing auditing when the system entry is performed by another user that can be identified as trustworthy.

  If the audit context is already initialized, the `PAM_REINITIALIZE_CRED` flag merges the current audit context with that of the user specified in `PAM_USER`. `PAM_REINITIALIZE_CRED` is useful when a user is assuming a new identity, as with `su(1M)`.

`PAM_DELETE_CRED`
: This flag has no effect and always returns `PAM_SUCCESS`.

The following options are interpreted:

`debug` Provides `syslog(3C)` debugging information at the `LOG_DEBUG` level.

`nowarn` Disables any warning messages.

Errors Upon successful completion of `pam_sm_setcred()`, `PAM_SUCCESS` is returned. The following error codes are returned upon error:

`PAM_CRED_UNAVAIL`	Underlying authentication service cannot retrieve user credentials
`PAM_CRED_EXPIRED`	User credentials have expired
`PAM_USER_UNKNOWN`	User is unknown to the authentication service
`PAM_CRED_ERR`	Failure in setting user credentials
`PAM_BUF_ERR`	Memory buffer error
`PAM_SYSTEM_ERR`	System error

The following values are returned from `pam_sm_authenticate()`:

`PAM_IGNORE`	Ignores this module regardless of the control flag

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

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

See Also `ssh(1)`, `su(1M)`, `settaskid(2)`, `libpam(3LIB)`, `getprojent(3PROJECT)`, `pam(3PAM)`, `pam_set_item(3PAM)`, `pam_sm_authenticate(3PAM)`, `syslog(3C)`, `setproject(3PROJECT)`,`pam.conf(4)`, `nsswitch.conf(4)`, `project(4)`, `attributes(5)`, `pam_authtok_check(5)`, `pam_authtok_get(5)`, `pam_authtok_store(5)`, `pam_dhkeys(5)`, `pam_passwd_auth(5)`, `pam_unix_auth(5)`, `pam_unix_account(5)`, `pam_unix_session(5)`, `privileges(5)`

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

If this module is replaced, the audit context and credential may not be correctly configured.

Name pam_unix_session – session management PAM module for UNIX

Synopsis `pam_unix_session.so.1`

Description The `pam_unix_session` module implements `pam_sm_open_session(3PAM)` and `pam_sm_close_session(3PAM)`.

`pam_sm_open_session()` updates the `/var/adm/lastlog` file with the information contained in the `PAM_USER`, `PAM_TTY`, and `PAM_RHOST` items. `pam_unix_account(5)` uses this account to determine the previous time the user logged in.

`pam_sm_close_session()` is a null function.

The following options can be passed to the module:

debug `syslog(3C)` debugging information at the `LOG_DEBUG` level

Errors Upon successful completion, `PAM_SUCCESS` is returned. The following error codes are returned upon error:

`PAM_SESSION_ERR` Cannot make or remove the entry for the specified session (PAM_TTY is not present).

`PAM_USER_UNKNOWN` No account is present for *user*.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT Level	MT-Safe with exceptions

See Also `pam(3PAM)`, `pam_authenticate(3PAM)`, `syslog(3C)`, `libpam(3LIB)`, `pam.conf(4)`, `nsswitch.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_account(5)`,`pam_unix_auth(5)`,

Notes The interfaces in `libpam(3LIB)` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

The `pam_unix`(5) module is no longer supported. 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 pkcs11_kernel – PKCS#11 interface to Kernel Cryptographic Framework

Synopsis

```
/usr/lib/security/pkcs11_kernel.so
/usr/lib/security/64/pkcs11_kernel.so
```

Description The `pkcs11_kernel.so` object implements the RSA PKCS#11 v2.20 specification by using a private interface to communicate with the Kernel Cryptographic Framework.

Each unique hardware provider is represented by a PKCS#11 slot. In a system with no hardware Kernel Cryptographic Framework providers, this PKCS#11 library presents no slots.

The PKCS#11 mechanisms provided by this library is determined by the available hardware providers.

Application developers should link to `libpkcs11.so` rather than link directly to `pkcs11_kernel.so`. See `libpkcs11(3LIB)`.

All of the Standard PKCS#11 functions listed on `libpkcs11(3LIB)` are implemented except for the following:

```
C_DecryptDigestUpdate
C_DecryptVerifyUpdate
C_DigestEncryptUpdate
C_GetOperationState
C_InitToken
C_InitPIN
C_SetOperationState
C_SignEncryptUpdate
C_WaitForSlotEvent
```

A call to these functions returns `CKR_FUNCTION_NOT_SUPPORTED`.

Buffers cannot be greater than 2 megabytes. For example, `C_Encrypt()` can be called with a 2 megabyte buffer of plaintext and a 2 megabyte buffer for the ciphertext.

The maximum number of object handles that can be returned by a call to `C_FindObjects()` is 512.

The maximum amount of kernel memory that can be used for crypto operations is limited by the `project.max-crypto-memory` resource control. Allocations in the kernel for buffers and session-related structures are charged against this resource control.

Return Values The return values of each of the implemented functions are defined and listed in the RSA PKCS#11 v2.20 specification. See http://www.rsasecurity.com.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard: PKCS#11 v2.20
MT-Level	MT-Safe with exceptions. See section 6.5.2 of RSA PKCS#11 v2.20

See Also `cryptoadm(1M)`, `rctladm(1M)`, `libpkcs11(3LIB)`, `attributes(5)`, `pkcs11_softtoken(5)`

RSA PKCS#11 v2.20 http://www.rsasecurity.com

Notes Applications that have an open session to a PKCS#11 slot make the corresponding hardware provider driver not unloadable. An administrator must close the applications that have an PKCS#11 session open to the hardware provider to make the driver unloadable.

Name pkcs11_softtoken – Software RSA PKCS#11 softtoken

Synopsis

```
/usr/lib/security/pkcs11_softtoken.so
/usr/lib/security/64/pkcs11_softtoken.so
```

Description The `pkcs11_softtoken.so` object implements the RSA PKCS#11 v2.20 specification in software. Persistent storage for "token" objects is provided by this PKCS#11 implementation.

Application developers should link to `libpkcs11.so` rather than link directly to `pkcs11_softtoken.so`. See `libpkcs11(3LIB)`.

The following cryptographic algorithms are implemented: DES, 3DES, AES, Blowfish, RC4, MD5, SHA1, SHA256, SHA384, SHA512, RSA, DSA, DH, and ECC.

All of the Standard PKCS#11 functions listed on `libpkcs11(3LIB)` are implemented except for the following:

```
C_GetObjectSize
C_InitPIN
C_InitToken
C_WaitForSlotEvent
```

A call to these functions returns `CKR_FUNCTION_NOT_SUPPORTED`.

The following RSA PKCS#11 v2.20 mechanisms are supported:

```
CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_RSA_PKCS
CKM_RSA_X_509

CKM_DSA_KEY_PAIR_GEN
CKM_DSA
CKM_DSA_SHA1

CKM_DH_PKCS_KEY_PAIR_GEN
CKM_DH_PKCS_DERIVE

CKM_EC_KEY_PAIR_GEN
CKM_ECDSA
CKM_ECDSA_SHA1
CKM_ECDH1_DERIVE

CKM_DES_KEY_GEN
CKM_DES_ECB
CKM_DES_CBC
CKM_DES_CBC_PAD

CKM_DES3_KEY_GEN
CKM_DES3_ECB
CKM_DES3_CBC
```

```
CKM_DES3_CBC_PAD

CKM_AES_KEY_GEN
CKM_AES_ECB
CKM_AES_CBC
CKM_AES_CBC_PAD
CKM_AES_CTR

CKM_BLOWFISH_KEY_GEN
CKM_BLOWFISH_CBC

CKM_RC4_KEY_GEN
CKM_RC4

CKM_MD5_RSA_PKCS
CKM_SHA1_RSA_PKCS
CKM_SHA256_RSA_PKCS
CKM_SHA384_RSA_PKCS
CKM_SHA512_RSA_PKCS

CKM_MD5
CKM_SHA_1
CKM_SHA256
CKM_SHA384
CKM_SHA512

CKM_MD5_HMAC
CKM_MD5_HMAC_GENERAL
CKM_SHA_1_HMAC
CKM_SHA_1_HMAC_GENERAL
CKM_SHA256_HMAC
CKM_SHA256_HMAC_GENERAL
CKM_SHA384_HMAC
CKM_SHA384_HMAC_GENERAL

CKM_MD5_KEY_DERIVATION
CKM_SHA1_KEY_DERIVATION
CKM_SHA256_KEY_DERIVATION
CKM_SHA384_KEY_DERIVATION
CKM_SHA512_KEY_DERIVATION

CKM_SSL3_PRE_MASTER_KEY_GEN
CKM_SSL3_MASTER_KEY_DERIVE
CKM_SSL3_KEY_AND_MAC_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE_DH
CKM_TLS_PRE_MASTER_KEY_GEN
CKM_TLS_MASTER_KEY_DERIVE
CKM_TLS_KEY_AND_MAC_DERIVE
CKM_TLS_MASTER_KEY_DERIVE_DH
```

Each of the following types of key objects has certain token-specific attributes that are set to true by default as a result of object creation, key/key pair generation, and key derivation.

Public key object	CKA_ENCRYPT, CKA_VERIFY, CKA_VERIFY_RECOVER
Private key object	CKA_DECRYPT, CKA_SIGN, CKA_SIGN_RECOVER, CKA_EXTRACTABLE
Secret key object	CKA_ENCRYPT, CKA_DECRYPT, CKA_SIGN, CKA_VERIFY, CKA_EXTRACTABLE

The following certificate objects are supported:

CKC_X_509	For CKC_X_509 certificate objects, the following attributes are supported: CKA_SUBJECT, CKA_VALUE, CKA_LABEL, CKA_ID, CKA_ISSUER, CKA_SERIAL_NUMBER, and CKA_CERTIFICATE_TYPE.
CKC_X_509_ATTR_CERT	For CKC_X_509_ATTR_CERT certificate objects, the following attributes are supported: CKA_OWNER, CKA_VALUE, CKA_LABEL, CKA_SERIAL_NUMBER, CKA_AC_ISSUER, CKA_ATTR_TYPES, and CKA_CERTIFICATE_TYPE.

The search operation of objects matching the template is performed at C_FindObjectsInit. The matched objects are cached for subsequent C_FindObjects operations.

The pkcs11_softtoken.so object provides a filesystem-based persistent token object store for storing token objects. The default location of the token object store is the user's home directory returned by getpwuid_r(). The user can override the default location by using the ${SOFTTOKEN_DIR} environment variable.

If the token object store has never been initialized, the C_Login() function might return CKR_OK but the user will not be able to create, generate, derive or find any private token object and receives CKR_PIN_EXPIRED.

The user must use the pktool(1) setpin command with the default passphrase “changeme” as the old passphrase to change the passphrase of the object store. This action is needed to initialize and set the passphrase to a newly created token object store.

After logging into object store with the new passphrase that was set by the pktool setpin command, the user can create and store the private token object in this newly created object store. Until the token object store is initialized by setpin, the C_Login() function is allowed, but all attempts by the user to create, generate, derive or find any private token object fails with a CKR_PIN_EXPIRED error.

The PIN provided for C_Login() and C_SetPIN() functions can be any string of characters with lengths between 1 and 256 and no embedded nulls.

Return Values The return values for each of the implemented functions are defined and listed in the RSA PKCS#11 v2.20 specification. See http://www.rsasecurity.com

Files

user_home_directory`/.sunw/pkcs11_softtoken`	user's default token object store
`${SOFTTOKEN_DIR}/pkcs11_softtoken`	alternate token object store

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with exceptions. See section 6.5.2 of RSA PKCS#11 v2.20.
Standard	PKCS#11 v2.20

See Also `pktool(1)`, `cryptoadm(1M)`, `libpkcs11(3LIB)`, `attributes(5)`, `pkcs11_kernel(5)`

RSA PKCS#11 v2.20 http://www.rsasecurity.com

Name pkcs11_tpm – RSA PKCS#11 token for Trusted Platform Modules (TPM)

Synopsis

```
/usr/lib/security/pkcs11_tpm.so

/usr/lib/security/64/pkcs11_tpm.so
```

Description The `pkcs11_tpm.so` object implements the RSA PKCS#11 v2.20 specification using Trusted Computing Group protocols to talk to a TPM security device. This provider implements the PKCS#11 specification and uses the TCG Software Stack (TSS) APIs in the `SUNWtrousers` package.

Application developers should link to `libpkcs11.so.1` rather than link directly with `pkcs11_tpm.so`. See `libpkcs11(3LIB)`.

The following cryptographic algorithms are implemented: `RSA`, `SHA1`, and `MD5`.

All of the standard PKCS#11 functions listed in `libpkcs11(3LIB)` are implemented except for the following:

```
C_EncryptUpdate
C_EncryptFinal
C_DecryptUpdate
C_DecryptFinal
C_DigestEncryptUpdate
C_DecryptDigestUpdate
C_SignEncryptUpdate
C_DecryptVerifyUpdate
C_GetFunctionStatus
C_CancelFunction
C_WaitForSlotEvent
C_GenerateKey
C_DeriveKey
```

The following RSA PKCS#11 v2.20 mechanisms are supported:

```
CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_RSA_PKCS
CKM_RSA_PKCS_OAEP
CKM_RSA_X_509
CKM_MD5_RSA_PKCS
CKM_SHA1_RSA_PKCS
CKM_SHA_1
CKM_SHA_1_HMAC
CKM_SHA_1_HMAC_GENERAL
CKM_MD5
CKM_MD5_HMAC
CKM_MD5_HMAC_GENERAL
```

Per-User Initialization

The `pkcs11_tpm` provider can only be used on a system which has a TPM device and which also has the `SUNWtrousers` package installed. If those prerequisites are met, users can create their own private tokens using `pktool(1)`, which will allow them to perform operations using the TPM device and protect their private data with TPM-protected keys.

To prepare and initialize a user's TPM token, the following steps must be performed:

1. Initialize the token.
2. Set the SO (security officer) PIN.
3. Set the user's unique PIN.

Initializing the token is done using the `pktool(1)` command as follows:

```
$ pktool inittoken currlabel=TPM newlabel=tpm/myname
```

- By default, an uninitialized TPM is recognized by the name `TPM`. When a user initializes their own private token, it can either be renamed to something else (for example, `tpm/joeuser`) or kept as `TPM` (in which case the `newlabel` argument would be omitted).
- The user will have to supply the default SO PIN before being able to initialize his or her token. The default SO PIN is `87654321`. It is changed in step 2, above.

Once the token is initialized, the SO and user PINs must be changed from the default values. Again, `pktool(1)` is used to change these PIN values.

Changing the SO PIN:

```
$ pktool setpin token=tpm/joeuser so
```

The `so` option indicates that this "setpin" operation is to change the SO PIN and must be present. The user must then enter the default SO PIN (`87654321`) and then enter (and confirm) a new PIN.

Once the SO PIN is reset from the default, the user's unique PIN must also be changed.

Changing the user's PIN:

```
$ pktool setpin token=tmp/joeuser
```

The default PIN for a non-SO user is `12345678`. The user must enter the default PIN and then enter (and confirm) a new, unique PIN.

The PIN provided for the `pktool setpin` operation or by calling `C_Login()` and `C_SetPIN()` functions can be any string of characters with a length between 1 and 256 and no embedded nulls.

Accessing the Token

After a user initializes their token, they can begin using it with `pktool(1)` or by writing PKCS11 applications and locating the token using the name created above (`tpm/joeuser` in the examples above).

Examples:

```
$ pktool gencert token=tpm/joeuser -i
$ pktool list token=tpm/joeuser
```

Notes

`pkcs11_tpm.so` provides object storage in a filesystem-specific token object storage area. Private objects are protected by encryption with private keys and can only be decrypted by loading the token's private key into the TPM and performing the decryption entirely in the TPM. The user's private key is generated by the TPM when the user sets their personal PIN (see above). The keys for both the SO and users are stored in the TSS persistent storage database and are referenced by a unique UUID value. All user tokens have a unique SO key and unique user key so that the PINs for one user's token will not unlock private data in another user's token on the same machine.

Each TPM is unique and the token keys created on one TPM may not be used on another TPM. The `pkcs11_tpm.so` token data is all managed on the system where the TPM resides and may not be moved to other systems. If the TPM is reset and the SRK (Storage Root Key) is changed, all of the keys previously generated for that TPM will no longer be valid.

`pkcs11_tpm.so` creates a private workspace to manage administrative files for each token created. By default, this area is created as `/var/tpm/pkcs11/$USERNAME`. However, users may override this by setting the `PKCS11_TPM_DIR` environment variable prior to initializing or using the token.

Return Values

The return values for each of the implemented functions are defined and listed in the RSA PKCS#11 v2.20 specification. See `http://www.rsasecurity.com`.

Files

`/var/tpm/pkcs11/USERNAME`
User's default token object store.

`${PKCS11_TPM_DIR}`
Alternate token object store.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with Exceptions (see below)
Standard	PKCS#11 v2.20

Exceptions to MT-Safe attribute are documented in section 6.5.2 of RSA PKCS#11 v2.20.

See Also

`pktool(1)`, `cryptoadm(1M)`, `libpkcs11(3LIB)`, `attributes(5)`

TCG Software Stack (TSS) Specifications: `https://www.trustedcomputinggroup.org/specs/TSS` (as of the date of publication)

Name privileges – process privilege model

Description Solaris software implements a set of privileges that provide fine-grained control over the actions of processes. The possession of a certain privilege allows a process to perform a specific set of restricted operations.

The change to a primarily privilege-based security model in the Solaris operating system gives developers an opportunity to restrict processes to those privileged operations actually needed instead of all (super-user) or no privileges (non-zero UIDs). Additionally, a set of previously unrestricted operations now requires a privilege; these privileges are dubbed the "basic" privileges and are by default given to all processes.

Taken together, all defined privileges with the exception of the "basic" privileges compose the set of privileges that are traditionally associated with the root user. The "basic" privileges are "privileges" unprivileged processes were accustomed to having.

The defined privileges are:

`PRIV_CONTRACT_EVENT`
Allow a process to request reliable delivery of events to an event endpoint.

Allow a process to include events in the critical event set term of a template which could be generated in volume by the user.

`PRIV_CONTRACT_IDENTITY`
Allows a process to set the service FMRI value of a process contract template.

`PRIV_CONTRACT_OBSERVER`
Allow a process to observe contract events generated by contracts created and owned by users other than the process's effective user ID.

Allow a process to open contract event endpoints belonging to contracts created and owned by users other than the process's effective user ID.

`PRIV_CPC_CPU`
Allow a process to access per-CPU hardware performance counters.

`PRIV_DTRACE_KERNEL`
Allow DTrace kernel-level tracing.

`PRIV_DTRACE_PROC`
Allow DTrace process-level tracing. Allow process-level tracing probes to be placed and enabled in processes to which the user has permissions.

`PRIV_DTRACE_USER`
Allow DTrace user-level tracing. Allow use of the syscall and profile DTrace providers to examine processes to which the user has permissions.

PRIV_FILE_CHOWN
Allow a process to change a file's owner user ID. Allow a process to change a file's group ID to one other than the process's effective group ID or one of the process's supplemental group IDs.

PRIV_FILE_CHOWN_SELF
Allow a process to give away its files. A process with this privilege runs as if {_POSIX_CHOWN_RESTRICTED} is not in effect.

PRIV_FILE_DAC_EXECUTE
Allow a process to execute an executable file whose permission bits or ACL would otherwise disallow the process execute permission.

PRIV_FILE_DAC_READ
Allow a process to read a file or directory whose permission bits or ACL would otherwise disallow the process read permission.

PRIV_FILE_DAC_SEARCH
Allow a process to search a directory whose permission bits or ACL would not otherwise allow the process search permission.

PRIV_FILE_DAC_WRITE
Allow a process to write a file or directory whose permission bits or ACL do not allow the process write permission. All privileges are required to write files owned by UID 0 in the absence of an effective UID of 0.

PRIV_FILE_DOWNGRADE_SL
Allow a process to set the sensitivity label of a file or directory to a sensitivity label that does not dominate the existing sensitivity label.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_FILE_LINK_ANY
Allow a process to create hardlinks to files owned by a UID different from the process's effective UID.

PRIV_FILE_OWNER
Allow a process that is not the owner of a file to modify that file's access and modification times. Allow a process that is not the owner of a directory to modify that directory's access and modification times. Allow a process that is not the owner of a file or directory to remove or rename a file or directory whose parent directory has the “save text image after execution” (sticky) bit set. Allow a process that is not the owner of a file to mount a `namefs` upon that file. Allow a process that is not the owner of a file or directory to modify that file's or directory's permission bits or ACL.

PRIV_FILE_SETID
Allow a process to change the ownership of a file or write to a file without the set-user-ID and set-group-ID bits being cleared. Allow a process to set the set-group-ID bit on a file or directory whose group is not the process's effective group or one of the process's

supplemental groups. Allow a process to set the set-user-ID bit on a file with different ownership in the presence of PRIV_FILE_OWNER. Additional restrictions apply when creating or modifying a setuid 0 file.

PRIV_FILE_UPGRADE_SL
: Allow a process to set the sensitivity label of a file or directory to a sensitivity label that dominates the existing sensitivity label.

  This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_FILE_FLAG_SET
: Allows a process to set immutable, nounlink or appendonly file attributes.

PRIV_GRAPHICS_ACCESS
: Allow a process to make privileged ioctls to graphics devices. Typically only an xserver process needs to have this privilege. A process with this privilege is also allowed to perform privileged graphics device mappings.

PRIV_GRAPHICS_MAP
: Allow a process to perform privileged mappings through a graphics device.

PRIV_IPC_DAC_READ
: Allow a process to read a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits would not otherwise allow the process read permission.

PRIV_IPC_DAC_WRITE
: Allow a process to write a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits would not otherwise allow the process write permission.

PRIV_IPC_OWNER
: Allow a process that is not the owner of a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment to remove, change ownership of, or change permission bits of the Message Queue, Semaphore Set, or Shared Memory Segment.

PRIV_NET_ACCESS
: Allow a process to open a TCP, UDP, SDP or SCTP network endpoint.

PRIV_NET_BINDMLP
: Allow a process to bind to a port that is configured as a multi-level port (MLP) for the process's zone. This privilege applies to both shared address and zone-specific address MLPs. See tnzonecfg(4) from the Trusted Extensions manual pages for information on configuring MLP ports.

  This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_NET_ICMPACCESS
: Allow a process to send and receive ICMP packets.

PRIV_NET_MAC_AWARE
Allow a process to set the NET_MAC_AWARE process flag by using setpflags(2). This privilege also allows a process to set the SO_MAC_EXEMPT socket option by using setsockopt(3SOCKET). The NET_MAC_AWARE process flag and the SO_MAC_EXEMPT socket option both allow a local process to communicate with an unlabeled peer if the local process's label dominates the peer's default label, or if the local process runs in the global zone.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_NET_OBSERVABILITY
Allow a process to open a device for just receiving network traffic, sending traffic is disallowed.

PRIV_NET_PRIVADDR
Allow a process to bind to a privileged port number. The privilege port numbers are 1-1023 (the traditional UNIX privileged ports) as well as those ports marked as "udp/tcp_extra_priv_ports" with the exception of the ports reserved for use by NFS and SMB.

PRIV_NET_RAWACCESS
Allow a process to have direct access to the network layer.

PRIV_PROC_AUDIT
Allow a process to generate audit records. Allow a process to get its own audit pre-selection information.

PRIV_PROC_CHROOT
Allow a process to change its root directory.

PRIV_PROC_CLOCK_HIGHRES
Allow a process to use high resolution timers.

PRIV_PROC_EXEC
Allow a process to call exec(2).

PRIV_PROC_FORK
Allow a process to call fork(2), fork1(2), or vfork(2).

PRIV_PROC_INFO
Allow a process to examine the status of processes other than those to which it can send signals. Processes that cannot be examined cannot be seen in /proc and appear not to exist.

PRIV_PROC_LOCK_MEMORY
Allow a process to lock pages in physical memory.

PRIV_PROC_OWNER
Allow a process to send signals to other processes and inspect and modify the process state in other processes, regardless of ownership. When modifying another process, additional restrictions apply: the effective privilege set of the attaching process must be a superset of

the target process's effective, permitted, and inheritable sets; the limit set must be a superset of the target's limit set; if the target process has any UID set to 0 all privilege must be asserted unless the effective UID is 0. Allow a process to bind arbitrary processes to CPUs.

`PRIV_PROC_PRIOCNTL`
Allow a process to elevate its priority above its current level. Allow a process to change its scheduling class to any scheduling class, including the RT class.

`PRIV_PROC_SESSION`
Allow a process to send signals or trace processes outside its session.

`PRIV_PROC_SETID`
Allow a process to set its UIDs at will, assuming UID 0 requires all privileges to be asserted.

`PRIV_PROC_TASKID`
Allow a process to assign a new task ID to the calling process.

`PRIV_PROC_ZONE`
Allow a process to trace or send signals to processes in other zones. See `zones(5)`.

`PRIV_SYS_ACCT`
Allow a process to enable and disable and manage accounting through `acct(2)`.

`PRIV_SYS_ADMIN`
Allow a process to perform system administration tasks such as setting node and domain name and specifying `coreadm(1M)` and `nscd(1M)` settings

`PRIV_SYS_AUDIT`
Allow a process to start the (kernel) audit daemon. Allow a process to view and set audit state (audit user ID, audit terminal ID, audit sessions ID, audit pre-selection mask). Allow a process to turn off and on auditing. Allow a process to configure the audit parameters (cache and queue sizes, event to class mappings, and policy options).

`PRIV_SYS_CONFIG`
Allow a process to perform various system configuration tasks. Allow filesystem-specific administrative procedures, such as filesystem configuration ioctls, quota calls, creation and deletion of snapshots, and manipulating the PCFS bootsector.

`PRIV_SYS_DEVICES`
Allow a process to create device special files. Allow a process to successfully call a kernel module that calls the kernel `drv_priv(9F)` function to check for allowed access. Allow a process to open the real console device directly. Allow a process to open devices that have been exclusively opened.

`PRIV_SYS_DL_CONFIG`
Allow a process to configure a system's datalink interfaces.

`PRIV_SYS_IP_CONFIG`
Allow a process to configure a system's IP interfaces and routes. Allow a process to configure network parameters for TCP/IP using `ndd`. Allow a process access to otherwise

restricted TCP/IP information using `ndd`. Allow a process to configure IPsec. Allow a process to pop anchored STREAMs modules with matching `zoneid`.

`PRIV_SYS_IPC_CONFIG`
Allow a process to increase the size of a System V IPC Message Queue buffer.

`PRIV_SYS_LINKDIR`
Allow a process to unlink and link directories.

`PRIV_SYS_MOUNT`
Allow a process to mount and unmount filesystems that would otherwise be restricted (that is, most filesystems except `namefs`). Allow a process to add and remove swap devices.

`PRIV_SYS_NET_CONFIG`
Allow a process to do all that `PRIV_SYS_IP_CONFIG`, `PRIV_SYS_DL_CONFIG`, and `PRIV_SYS_PPP_CONFIG` allow, plus the following: use the `rpcmod` STREAMS module and insert/remove STREAMS modules on locations other than the top of the module stack.

`PRIV_SYS_NFS`
Allow a process to provide NFS service: start NFS kernel threads, perform NFS locking operations, bind to NFS reserved ports: ports 2049 (`nfs`) and port 4045 (`lockd`).

`PRIV_SYS_PPP_CONFIG`
Allow a process to create, configure, and destroy PPP instances with pppd(1M) `pppd(1M)` and control PPPoE plumbing with `sppptun(1M)`sppptun(1M). This privilege is granted by default to exclusive IP stack instance zones.

`PRIV_SYS_RES_CONFIG`
Allow a process to create and delete processor sets, assign CPUs to processor sets and override the `PSET_NOESCAPE` property. Allow a process to change the operational status of CPUs in the system using `p_online(2)`. Allow a process to configure filesystem quotas. Allow a process to configure resource pools and bind processes to pools.

`PRIV_SYS_RESOURCE`
Allow a process to exceed the resource limits imposed on it by `setrlimit(2)` and `setrctl(2)`.

`PRIV_SYS_SMB`
Allow a process to provide NetBIOS or SMB services: start SMB kernel threads or bind to NetBIOS or SMB reserved ports: ports 137, 138, 139 (NetBIOS) and 445 (SMB).

`PRIV_SYS_SUSER_COMPAT`
Allow a process to successfully call a third party loadable module that calls the kernel `suser()` function to check for allowed access. This privilege exists only for third party loadable module compatibility and is not used by Solaris proper.

`PRIV_SYS_TIME`
Allow a process to manipulate system time using any of the appropriate system calls: `stime(2)`, `adjtime(2)`, and `ntp_adjtime(2)`.

PRIV_SYS_TRANS_LABEL
Allow a process to translate labels that are not dominated by the process's sensitivity label to and from an external string form.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_VIRT_MANAGE
Allows a process to manage virtualized environments such as xVM(5).

PRIV_WIN_COLORMAP
Allow a process to override colormap restrictions.

Allow a process to install or remove colormaps.

Allow a process to retrieve colormap cell entries allocated by other processes.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_WIN_CONFIG
Allow a process to configure or destroy resources that are permanently retained by the X server.

Allow a process to use SetScreenSaver to set the screen saver timeout value

Allow a process to use ChangeHosts to modify the display access control list.

Allow a process to use GrabServer.

Allow a process to use the SetCloseDownMode request that can retain window, pixmap, colormap, property, cursor, font, or graphic context resources.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_WIN_DAC_READ
Allow a process to read from a window resource that it does not own (has a different user ID).

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_WIN_DAC_WRITE
Allow a process to write to or create a window resource that it does not own (has a different user ID). A newly created window property is created with the window's user ID.

This privilege is interpreted only if the system is configured with Trusted Extensions.

PRIV_WIN_DEVICES
Allow a process to perform operations on window input devices.

Allow a process to get and set keyboard and pointer controls.

Allow a process to modify pointer button and key mappings.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_DGA`
Allow a process to use the direct graphics access (DGA) X protocol extensions. Direct process access to the frame buffer is still required. Thus the process must have MAC and DAC privileges that allow access to the frame buffer, or the frame buffer must be allocated to the process.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_DOWNGRADE_SL`
Allow a process to set the sensitivity label of a window resource to a sensitivity label that does not dominate the existing sensitivity label.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_FONTPATH`
Allow a process to set a font path.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_MAC_READ`
Allow a process to read from a window resource whose sensitivity label is not equal to the process sensitivity label.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_MAC_WRITE`
Allow a process to create a window resource whose sensitivity label is not equal to the process sensitivity label. A newly created window property is created with the window's sensitivity label.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_SELECTION`
Allow a process to request inter-window data moves without the intervention of the selection confirmer.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_WIN_UPGRADE_SL`
Allow a process to set the sensitivity label of a window resource to a sensitivity label that dominates the existing sensitivity label.

This privilege is interpreted only if the system is configured with Trusted Extensions.

`PRIV_XVM_CONTROL`
Allows a process access to the `xVM(5)` control devices for managing guest domains and the hypervisor. This privilege is used only if booted into xVM on x86 platforms.

Of the privileges listed above, the privileges PRIV_FILE_LINK_ANY, PRIV_PROC_INFO, PRIV_PROC_SESSION, PRIV_NET_ACCESS, PRIV_PROC_FORK, and PRIV_PROC_EXEC are considered "basic" privileges. These are privileges that used to be always available to unprivileged processes. By default, processes still have the basic privileges.

The privileges PRIV_PROC_SETID and PRIV_PROC_AUDIT must be present in the Limit set (see below) of a process in order for setuid root execs to be successful; that is, get an effective UID of 0 and additional privileges.

The privilege implementation in Solaris extends the process credential with four privilege sets:

I, the inheritable set	The privileges inherited on exec.
P, the permitted set	The maximum set of privileges for the process.
E, the effective set	The privileges currently in effect.
L, the limit set	The upper bound of the privileges a process and its offspring can obtain. Changes to L take effect on the next exec.

The sets I, P and E are typically identical to the basic set of privileges for unprivileged processes. The limit set is typically the full set of privileges.

Each process has a Privilege Awareness State (PAS) that can take the value PA (privilege-aware) and NPA (not-PA). PAS is a transitional mechanism that allows a choice between full compatibility with the old superuser model and completely ignoring the effective UID.

To facilitate the discussion, we introduce the notion of "observed effective set" (oE) and "observed permitted set" (oP) and the implementation sets iE and iP.

A process becomes privilege-aware either by manipulating the effective, permitted, or limit privilege sets through setppriv(2) or by using setpflags(2). In all cases, oE and oP are invariant in the process of becoming privilege-aware. In the process of becoming privilege-aware, the following assignments take place:

```
iE = oE
iP = oP
```

When a process is privilege-aware, oE and oP are invariant under UID changes. When a process is not privilege-aware, oE and oP are observed as follows:

```
oE = euid == 0 ? L : iE
oP = (euid == 0 || ruid == 0 || suid == 0) ? L : iP
```

When a non-privilege-aware process has an effective UID of 0, it can exercise the privileges contained in its limit set, the upper bound of its privileges. If a non-privilege-aware process has any of the UIDs 0, it appears to be capable of potentially exercising all privileges in L.

It is possible for a process to return to the non-privilege aware state using `setpflags()`. The kernel always attempts this on exec(2). This operation is permitted only if the following conditions are met:

- If any of the UIDs is equal to 0, P must be equal to L.
- If the effective UID is equal to 0, E must be equal to L.

When a process gives up privilege awareness, the following assignments take place:

```
if (euid == 0) iE = L & I
if (any uid == 0) iP = L & I
```

The privileges obtained when not having a UID of `0` are the inheritable set of the process restricted by the limit set.

Only privileges in the process's (observed) effective privilege set allow the process to perform restricted operations. A process can use any of the privilege manipulation functions to add or remove privileges from the privilege sets. Privileges can be removed always. Only privileges found in the permitted set can be added to the effective and inheritable set. The limit set cannot grow. The inheritable set can be larger than the permitted set.

When a process performs an exec(2), the kernel first tries to relinquish privilege awareness before making the following privilege set modifications:

```
E' = P' = I' = L & I
L is unchanged
```

If a process has not manipulated its privileges, the privilege sets effectively remain the same, as E, P and I are already identical.

The limit set is enforced at `exec` time.

To run a non-privilege-aware application in a backward-compatible manner, a privilege-aware application should start the non-privilege-aware application with I=basic.

For most privileges, absence of the privilege simply results in a failure. In some instances, the absense of a privilege can cause system calls to behave differently. In other instances, the removal of a privilege can force a set-uid application to seriously malfunction. Privileges of this type are considered “unsafe”. When a process is lacking any of the unsafe privileges from its limit set, the system does not honor the set-uid bit of set-uid root applications. The following unsafe privileges have been identified: `proc_setid`, `sys_resource` and `proc_audit`.

Privilege Escalation

In certain circumstances, a single privilege could lead to a process gaining one or more additional privileges that were not explicitly granted to that process. To prevent such an escalation of privileges, the security policy requires explicit permission for those additional privileges.

Common examples of escalation are those mechanisms that allow modification of system resources through “raw" interfaces; for example, changing kernel data structures through `/dev/kmem` or changing files through `/dev/dsk/*`. Escalation also occurs when a process controls processes with more privileges than the controlling process. A special case of this is manipulating or creating objects owned by UID 0 or trying to obtain UID 0 using `setuid(2)`. The special treatment of UID 0 is needed because the UID 0 owns all system configuration files and ordinary file protection mechanisms allow processes with UID 0 to modify the system configuration. With appropriate file modifications, a given process running with an effective UID of 0 can gain all privileges.

In situations where a process might obtain UID 0, the security policy requires additional privileges, up to the full set of privileges. Such restrictions could be relaxed or removed at such time as additional mechanisms for protection of system files became available. There are no such mechanisms in the current Solaris release.

The use of UID 0 processes should be limited as much as possible. They should be replaced with programs running under a different UID but with exactly the privileges they need.

Daemons that never need to `exec` subprocesses should remove the `PRIV_PROC_EXEC` privilege from their permitted and limit sets.

Assigned Privileges and Safeguards

When privileges are assigned to a user, the system administrator could give that user more powers than intended. The administrator should consider whether safeguards are needed. For example, if the `PRIV_PROC_LOCK_MEMORY` privilege is given to a user, the administrator should consider setting the `project.max-locked-memory` resource control as well, to prevent that user from locking all memory.

Privilege Debugging

When a system call fails with a permission error, it is not always immediately obvious what caused the problem. To debug such a problem, you can use a tool called *privilege debugging*. When privilege debugging is enabled for a process, the kernel reports missing privileges on the controlling terminal of the process. (Enable debugging for a process with the `-D` option of `ppriv(1)`.) Additionally, the administrator can enable system-wide privilege debugging by setting the `system(4)` variable `priv_debug` using:

```
set priv_debug = 1
```

On a running system, you can use `mdb(1)` to change this variable.

Privilege Administration

The Solaris Management Console (see `smc(1M)`) is the preferred method of modifying privileges for a command. Use `usermod(1M)` or `smrole(1M)` to assign privileges to or modify privileges for, respectively, a user or a role. Use `ppriv(1)` to enumerate the privileges supported on a system and `truss(1)` to determine which privileges a program requires.

See Also `mdb(1)`, `ppriv(1)`, `add_drv(1M)`, `ifconfig(1M)`, `lockd(1M)`, `nfsd(1M)`, `pppd(1M)`, `rem_drv(1M)`, `smbd(1M)`, `sppptun(1M)`, `update_drv(1M)`, `Intro(2)`, `access(2)`, `acct(2)`, `acl(2)`, `adjtime(2)`, `audit(2)`, `auditon(2)`, `chmod(2)`, `chown(2)`, `chroot(2)`, `creat(2)`, `exec(2)`,

fcntl(2), fork(2), fpathconf(2), getacct(2), getpflags(2), getppriv(2), getsid(2), kill(2), link(2), memcntl(2), mknod(2), mount(2), msgctl(2), nice(2), ntp_adjtime(2), open(2), p_online(2), priocntl(2), priocntlset(2), processor_bind(2), pset_bind(2), pset_create(2), readlink(2), resolvepath(2), rmdir(2), semctl(2), setauid(2), setegid(2), seteuid(2), setgid(2), setgroups(2), setpflags(2), setppriv(2), setrctl(2), setregid(2), setreuid(2), setrlimit(2), settaskid(2), setuid(2), shmctl(2), shmget(2), shmop(2), sigsend(2), stat(2), statvfs(2), stime(2), swapctl(2), sysinfo(2), uadmin(2), ulimit(2), umount(2), unlink(2), utime(2), utimes(2), bind(3SOCKET), door_ucred(3C), priv_addset(3C), priv_set(3C), priv_getbyname(3C), priv_getbynum(3C), priv_set_to_str(3C), priv_str_to_set(3C), socket(3SOCKET), t_bind(3NSL), timer_create(3C), ucred_get(3C), exec_attr(4), proc(4), system(4), user_attr(4), xVM(5), ddi_cred(9F), drv_priv(9F), priv_getbyname(9F), priv_policy(9F), priv_policy_choice(9F), priv_policy_only(9F)

System Administration Guide: Security Services

Notes Removal of any of the basic privileges from a process leaves it in a non-standards compliant state, may cause unexpected application failures, and should only be performed with full knowledge of the potential side effects.

Name prof – profile within a function

Synopsis

```
#define MARK
#include <prof.h>

void MARK(name);
```

Description MARK introduces a mark called *name* that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of letters, numbers, or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header `prof.h` is included, either by a preprocessor directive as in the synopsis, or by a command line argument:

```
cc –p –DMARK work.c
```

If MARK is not defined, the MARK(*name*) statements may be left in the source files containing them and are ignored. `prof` –g must be used to get information on all labels.

Examples In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with MARK defined on the command line, the marks are ignored.

```
#include <prof.h>
work( )
{
        int i, j;
        . . .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

See Also profil(2), monitor(3C)

Name rbac, RBAC – role-based access control

Description The addition of role-based access control (RBAC) to the Solaris operating environment gives developers the opportunity to deliver fine-grained security in new and modified applications. RBAC is an alternative to the all-or-nothing security model of traditional superuser-based systems. With RBAC, an administrator can assign privileged functions to specific user accounts (or special accounts called roles).

There are two ways to give applications privileges:

1. Administrators can assign special attributes such as setUID to application binaries (executable files).
2. Administrators can assign special attributes such as setUID to applications using execution profiles.

Special attribute assignment along with the theory behind RBAC is discussed in detail in "Role Based Access Control" chapter of the *System Administration Guide: Security Services*. This chapter describes what authorizations are and how to code for them.

Authorizations An authorization is a unique string that represents a user's right to perform some operation or class of operations. Authorization definitions are stored in a database called `auth_attr(4)`. For programming authorization checks, only the authorization name is significant.

Some typical values in an `auth_attr` database are shown below.

```
solaris.jobs.:::Cron and At Jobs::help=JobHeader.html
solaris.jobs.grant:::Delegate Cron & At \
    Administration::help=JobsGrant.html
solaris.jobs.admin:::Manage All Jobs::help=AuthJobsAdmin.html
solaris.jobs.user:::Cron & At User::help=JobsUser.html
```

Authorization name strings ending with the `grant` suffix are special authorizations that give a user the ability to delegate authorizations with the same prefix and functional area to other users.

Creating Authorization Checks To check authorizations, use the `chkauthattr(3SECDB)` library function, which verifies whether or not a user has a given authorization. The synopsis is:

```
int chkauthattr(const char *authname, const char *username);
```

The `chkauthattr()` function checks the `policy.conf(4)`, `user_attr(4)`, and `prof_attr(4)` databases in order for a match to the given authorization.

If you are modifying existing code that tests for root UID, you should find the test in the code and replace it with the `chkauthattr()` function. A typical root UID check is shown in the first

code segment below. An authorization check replacing it is shown in the second code segment; it uses the `solaris.jobs.admin` authorization and a variable called `real_login` representing the user.

EXAMPLE 1 Standard root check

```
ruid = getuid();

if ((eflag || lflag || rflag) && argc == 1) {
        if ((pwp = getpwnam(*argv)) == NULL)
                crabort(INVALIDUSER);

        if (ruid != 0) {
                if (pwp->pw_uid != ruid)
                        crabort(NOTROOT);
                else
                        pp = getuser(ruid);
        } else
                pp = *argv++;
} else {
```

EXAMPLE 2 Authorization check

```
ruid = getuid();
if ((pwp = getpwuid(ruid)) == NULL)
        crabort(INVALIDUSER);

strcpy(real_login, pwp->pw_name);

if ((eflag || lflag || rflag) && argc == 1) {
        if ((pwp = getpwnam(*argv)) == NULL)
                crabort(INVALIDUSER);

        if (!chkauthattr("solaris.jobs.admin", real_login)) {
                if (pwp->pw_uid != ruid)
                        crabort(NOTROOT);
                else
                        pp = getuser(ruid);
        } else
                pp = *argv++;
} else {
```

For new applications, find an appropriate location for the test and use `chkauthattr()` as shown above. Typically the authorization check makes an access decision based on the identity of the calling user to determine if a privileged action (for example, a system call) should be taken on behalf of that user.

Applications that perform a test to restrict who can perform their security-relevant functionality are generally `setuid` to root. Programs that were written prior to RBAC and that are only available to the root user may not have such checks. In most cases, the kernel requires an effective user ID of root to override policy enforcement. Therefore, authorization checking is most useful in programs that are `setuid` to root.

For instance, if you want to write a program that allows authorized users to set the system date, the command must be run with an effective user ID of root. Typically, this means that the file modes for the file would be `-rwsr-xr-x` with root ownership.

Use caution, though, when making programs `setuid` to root. For example, the effective UID should be set to the real UID as early as possible in the program's initialization function. The effective UID can then be set back to root after the authorization check is performed and before the system call is made. On return from the system call, the effective UID should be set back to the real UID again to adhere to the principle of least privilege.

Another consideration is that `LD_LIBRARY` path is ignored for setuid programs (see SECURITY section in `ld.so.1(1)`) and that shell scripts must be modified to work properly when the effective and real UIDs are different. For example, the `-p` flag in Bourne shell is required to avoid resetting the effective UID back to the real UID.

Using an effective UID of root instead of the real UID requires extra care when writing shell scripts. For example, many shell scripts check to see if the user is root before executing their functionality. With RBAC, these shell scripts may be running with the effective UID of root and with a real UID of a user or role. Thus, the shell script should check `euid` instead of `uid`. For example,

```
WHO=‘id | cut -f1 -d" "‘
if [ ! "$WHO" = "uid=0(root)" ]
then
       echo "$PROG: ERROR: you must be super-user to run this script."
       exit 1
fi
```

should be changed to

```
WHO=‘/usr/xpg4/bin/id -n -u‘
if [ ! "$WHO" = "root" ]
then
       echo "$PROG: ERROR: you are not authorized to run this script."
       exit 1
fi
```

Authorizations can be explicitly checked in shell scripts by checking the output of the `auths(1)` utility. For example,

```
for auth in ‘auths | tr , " "‘ NOTFOUND
do
```

```
    [ "$auth" = "solaris.date" ] && break       # authorization found
done

if [ "$auth" != "solaris.date" ]
then
    echo >&2 "$PROG: ERROR: you are not authorized to set the date"
    exit 1
fi
```

See Also ld.so.1(1), chkauthattr(3SECDB), auth_attr(4), policy.conf(4), prof_attr(4), user_attr(4)

System Administration Guide: Security Services

Name regex – internationalized basic and extended regular expression matching

Description Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings. The Internationalized Regular Expressions described below differ from the Simple Regular Expressions described on the regexp(5) manual page in the following ways:

- both Basic and Extended Regular Expressions are supported
- the Internationalization features—character class, equivalence class, and multi-character collation—are supported.

The Basic Regular Expression (BRE) notation and construction rules described in the BASIC REGULAR EXPRESSIONS section apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in the EXTENDED REGULAR EXPRESSIONS section; any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interfaces regcomp(3C) and regexec(3C).

Basic Regular Expressions

BREs Matching a Single Character

A BRE ordinary character, a special character preceded by a backslash, or a period matches a single character. A bracket expression matches a single character or a single collating element. See RE Bracket Expression, below.

BRE Ordinary Characters

An ordinary character is a BRE that matches itself: any character in the supported character set, except for the BRE special characters listed in BRE Special Characters, below.

The interpretation of an ordinary character preceded by a backslash (\) is undefined, except for:

1. the characters), (, {, and }
2. the digits 1 to 9 inclusive (see BREs Matching Multiple Characters, below)
3. a character inside a bracket expression.

BRE Special Characters

A BRE *special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character will be a BRE that matches the special character itself. The BRE special characters and the contexts in which they have their special meaning are:

. [\ The period, left-bracket, and backslash are special except when used in a bracket expression (see RE Bracket Expression, below). An expression containing a [that is not preceded by a backslash and is not part of a bracket expression produces undefined results.

* The asterisk is special except when used:

- in a bracket expression
- as the first character of an entire BRE (after an initial ^, if any)

- as the first character of a subexpression (after an initial ^, if any); see BREs Matching Multiple Characters, below.

^ The circumflex is special when used:

- as an anchor (see BRE Expression Anchoring, below).
- as the first character of a bracket expression (see RE Bracket Expression, below).

$ The dollar sign is special when used as an anchor.

Periods in BREs A period (.), when used outside a bracket expression, is a BRE that matches any character in the supported character set except NUL.

RE Bracket Expression A bracket expression (an expression enclosed in square brackets, [ ]) is an RE that matches a single collating element contained in the non-empty set of collating elements represented by the bracket expression.

The following rules and definitions apply to bracket expressions:

1. A *bracket expression* is either a matching list expression or a non-matching list expression. It consists of one or more expressions: collating elements, collating symbols, equivalence classes, character classes, or range expressions (see rule 7 below). Portable applications must not use range expressions, even though all implementations support them. The right-bracket (]) loses its special meaning and represents itself in a bracket expression if it occurs first in the list (after an initial circumflex (^), if any). Otherwise, it terminates the bracket expression, unless it appears in a collating symbol (such as [.].]) or is the ending right-bracket for a collating symbol, equivalence class, or character class. The special characters:

   ```
   .   *   [   \
   ```

   (period, asterisk, left-bracket and backslash, respectively) lose their special meaning within a bracket expression.

   The character sequences:

   ```
   [.   [=   [:
   ```

   (left-bracket followed by a period, equals-sign, or colon) are special inside a bracket expression and are used to delimit collating symbols, equivalence class expressions, and character class expressions. These symbols must be followed by a valid expression and the matching terminating sequence .], =] or :], as described in the following items.
2. A *matching list* expression specifies a list that matches any one of the expressions represented in the list. The first character in the list must not be the circumflex. For example, `[abc]` is an RE that matches any of the characters `a`, `b` or `c`.
3. A *non-matching list* expression begins with a circumflex (^), and specifies a list that matches any character or collating element except for the expressions represented in the list after the leading circumflex. For example, [^abc] is an RE that matches any character or

collating element except the characters `a`, `b`, or `c`. The circumflex will have this special meaning only when it occurs first in the list, immediately following the left-bracket.

4. A *collating symbol* is a collating element enclosed within bracket-period ([..]) delimiters. Multi-character collating elements must be represented as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multi-character collating element. For example, if the string `ch` is a collating element in the current collation sequence with the associated collating symbol <ch>, the expression [[.ch.]] will be treated as an RE matching the character sequence `ch`, while [ch] will be treated as an RE matching `c` or `h`. Collating symbols will be recognized only inside bracket expressions. This implies that the RE `[[.ch.]]*c` matches the first to fifth character in the string chchch. If the string is not a collating element in the current collating sequence definition, or if the collating element has no characters associated with it, the symbol will be treated as an invalid expression.

5. An *equivalence class expression* represents the set of collating elements belonging to an equivalence class. Only primary equivalence classes will be recognised. The class is expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal ([==]) delimiters. For example, if a and b belong to the same equivalence class, then [[=a=]b], [[==]b] and [[==]b] will each be equivalent to [ab]. If the collating element does not belong to an equivalence class, the equivalence class expression will be treated as a *collating symbol*.

6. A *character class expression* represents the set of characters belonging to a character class, as defined in the `LC_CTYPE` category in the current locale. All character classes specified in the current locale will be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon ([::]) delimiters.

   The following character class expressions are supported in all locales:

`[:alnum:]`	`[:cntrl:]`	`[:lower:]`	`[:space:]`
`[:alpha:]`	`[:digit:]`	`[:print:]`	`[:upper:]`
`[:blank:]`	`[:graph:]`	`[:punct:]`	`[:xdigit:]`

   In addition, character class expressions of the form:

   ```
   [:name:]
   ```

   are recognized in those locales where the *name* keyword has been given a `charclass` definition in the `LC_CTYPE` category.

7. A *range expression* represents the set of collating elements that fall between two elements in the current collation sequence, inclusively. It is expressed as the starting point and the ending point separated by a hyphen (–).

Range expressions must not be used in portable applications because their behavior is dependent on the collating sequence. Ranges will be treated according to the current collating sequence, and include such characters that fall within the range based on that collating sequence, regardless of character values. This, however, means that the interpretation will differ depending on collating sequence. If, for instance, one collating sequence defines as a variant of a, while another defines it as a letter following z, then the expression [–z] is valid in the first language and invalid in the second.

In the following, all examples assume the collation sequence specified for the POSIX locale, unless another collation sequence is specifically defined.

The starting range point and the ending range point must be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. For example, the unspecified expression [[=e=]–f] should be given as [[=e=]e–f]. The ending range point must collate equal to or higher than the starting range point; otherwise, the expression will be treated as invalid. The order used is the order in which the collating elements are specified in the current collation definition. One-to-many mappings (see `locale(5)`) will not be performed. For example, assuming that the character eszet is placed in the collation sequence after `r` and `s`, but before `t`, and that it maps to the sequence `ss` for collation purposes, then the expression [r–s] matches only `r` and `s`, but the expression [s–t] matches `s`, `beta`, or `t`.

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for instance [a–m–o]) is undefined.

The hyphen character will be treated as itself if it occurs first (after an initial ^, if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions [–ac] and [ac–] are equivalent and match any of the characters a, c, or –; [^–ac] and [^ac–] are equivalent and match any characters except a, c, or –; the expression [%– –] matches any of the characters between % and – inclusive; the expression [– –@] matches any of the characters between – and @ inclusive; and the expression [a– –@] is invalid, because the letter a follows the symbol – in the POSIX locale. To use a hyphen as the starting range point, it must either come first in the bracket expression or be specified as a collating symbol, for example: [][.–.]–0], which matches either a right bracket or any character or collating element that collates between hyphen and 0, inclusive.

If a bracket expression must specify both – and], the] must be placed first (after the ^, if any) and the – last within the bracket expression.

Note: Latin-1 characters such as or are not printable in some locales, for example, the `ja` locale.

BREs Matching Multiple Characters

The following rules can be used to construct BREs matching multiple characters from BREs matching a single character:

1. The concatenation of BREs matches the concatenation of the strings matched by each component of the BRE.

2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs \(and \) . Such a subexpression matches whatever it would have matched without the \(and \), except that anchoring within subexpressions is optional behavior; see `BRE Expression Anchoring`, below. Subexpressions can be arbitrarily nested.
3. The *back-reference* expression *n* matches the same (possibly empty) string of characters as was matched by a subexpression enclosed between \(and \) preceding the *n*. The character *n* must be a digit from 1 to 9 inclusive, *n*th subexpression (the one that begins with the *n*th \(and ends with the corresponding paired \)). The expression is invalid if less than *n* subexpressions precede the *n*. For example, the expression `^\(.*\)\1$` matches a line consisting of two adjacent appearances of the same string, and the expression `\(a\)*\1` fails to match `a`. The limit of nine back-references to subexpressions in the RE is based on the use of a single digit identifier. This does not imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten subexpressions:

   ```
   \(\(\(ab\)*c\)*d\)\(ef\)*\(gh\)\{2\}\(ij\)*\(kl\)*\(mn\)*\(op\)*\(qr\)*
   ```

4. When a BRE matching a single character, a subexpression or a back-reference is followed by the special character asterisk (*), together with that asterisk it matches what zero or more consecutive occurrences of the BRE would match. For example, `[ab]*` and `[ab][ab]` are equivalent when matching the string `ab`.
5. When a BRE matching a single character, a subexpression, or a back-reference is followed by an *interval expression* of the format \{*m*\}, \{*m*,\} or \{*m*,*n*\}, together with that interval expression it matches what repeated consecutive occurrences of the BRE would match. The values of *m* and *n* will be decimal integers in the range $0 \le m \le n \le$ {`RE_DUP_MAX`}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression \{*m*\} matches exactly *m* occurrences of the preceding BRE, \{*m*,\} matches at least *m* occurrences and \{*m*,*n*\} matches any number of occurrences between *m* and *n*, inclusive.

   For example, in the string `abababccccccd`, the BRE c\{3\} is matched by characters seven to nine, the BRE \(ab\)\{4,\} is not matched at all and the BRE c\{1,3\}d is matched by characters ten to thirteen.

The behavior of multiple adjacent duplication symbols (* and intervals) produces undefined results.

BRE Precedence

The order of precedence is as shown in the following table:

BRE Precedence (from high to low)	
collation-related bracket symbols	[= =] [: :] [. .]
escaped characters	\<*special character*>
bracket expression	[ ]
subexpressions/back-references	\(\) *n*

single-character-BRE duplication	* \{*m*,*n*\}
concatenation	
anchoring	^ $

BRE Expression Anchoring

A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters will be considered BRE anchors in the following contexts:

1. A circumflex (`^`) is an anchor when used as the first character of an entire BRE. The implementation may treat circumflex as an anchor when used as the first character of a subexpression. The circumflex will anchor the expression to the beginning of a string; only sequences starting at the first character of a string will be matched by the BRE. For example, the BRE `^ab` matches `ab` in the string `abcdef`, but fails to match in the string `cdefab`. A portable BRE must escape a leading circumflex in a subexpression to match a literal circumflex.
2. A dollar sign (`$`) is an anchor when used as the last character of an entire BRE. The implementation may treat a dollar sign as an anchor when used as the last character of a subexpression. The dollar sign will anchor the expression to the end of the string being matched; the dollar sign can be said to match the end-of-string following the last character.
3. A BRE anchored by both `^` and `$` matches only an entire string. For example, the BRE `^abcdef$` matches strings consisting only of `abcdef`.
4. `^` and `$` are not special in subexpressions.

Note: The Solaris implementation does not support anchoring in BRE subexpressions.

Extended Regular Expressions

The rules specifiifed for BREs apply to Extended Regular Expressions (EREs) with the following exceptions:

- The characters `|`, `+`, and `?` have special meaning, as defined below.
- The `{` and `}` characters, when used as the duplication operator, are not preceded by backslashes. The constructs `\{` and `\}` simply match the characters `{` and `}`, respectively.
- The back reference operator is not supported.
- Anchoring (`^$`) is supported in subexpressions.

EREs Matching a Single Character

An ERE ordinary character, a special character preceded by a backslash, or a period matches a single character. A bracket expression matches a single character or a single collating element. An *ERE matching a single character* enclosed in parentheses matches the same as the ERE without parentheses would have matched.

ERE Ordinary Characters

An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in `ERE Special Characters` below. The interpretation of an ordinary character preceded by a backslash (`\`) is undefined.

ERE Special Characters

An *ERE special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character is an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they have their special meaning are:

. [\ (
: The period, left-bracket, backslash, and left-parenthesis are special except when used in a bracket expression (see RE `Bracket Expression`, above). Outside a bracket expression, a left-parenthesis immediately followed by a right-parenthesis produces undefined results.

)
: The right-parenthesis is special when matched with a preceding left-parenthesis, both outside a bracket expression.

* + ? {
: The asterisk, plus-sign, question-mark, and left-brace are special except when used in a bracket expression (see RE `Bracket Expression`, above). Any of the following uses produce undefined results:

  - if these characters appear first in an ERE, or immediately following a vertical-line, circumflex or left-parenthesis
  - if a left-brace is not part of a valid interval expression.

|
: The vertical-line is special except when used in a bracket expression (see RE `Bracket Expression`, above). A vertical-line appearing first or last in an ERE, or immediately following a vertical-line or a left-parenthesis, or immediately preceding a right-parenthesis, produces undefined results.

^
: The circumflex is special when used:

  - as an anchor (see ERE `Expression Anchoring`, below).
  - as the first character of a bracket expression (see RE `Bracket Expression`, above).

$
: The dollar sign is special when used as an anchor.

Periods in EREs

A period (`.`), when used outside a bracket expression, is an ERE that matches any character in the supported character set except NUL.

ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see RE `Bracket Expression`, above).

EREs Matching Multiple Characters

The following rules will be used to construct EREs matching multiple characters from EREs matching a single character:

1. A *concatenation of EREs* matches the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses matches whatever the concatenation without the parentheses matches. For example, both the ERE `cd` and the ERE `(cd)` are matched by the third and fourth character of the string `abcdefabcdef`.

2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character plus-sign (+), together with that plus-sign it matches what one or more consecutive occurrences of the ERE would match. For example, the ERE `b+(bc)` matches the fourth to seventh characters in the string `acabbbcde`; `[ab]+` and `[ab][ab]*` are equivalent.
3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character asterisk (*), together with that asterisk it matches what zero or more consecutive occurrences of the ERE would match. For example, the ERE `b*c` matches the first character in the string `cabbbcde`, and the ERE `b*cd` matches the third to seventh characters in the string `cabbbcdebbbbbbcdbc`. And, `[ab]*` and `[ab][ab]` are equivalent when matching the string `ab`.
4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character question-mark (?), together with that question-mark it matches what zero or one consecutive occurrences of the ERE would match. For example, the ERE `b?c` matches the second character in the string `acabbbcde`.
5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by an *interval expression* of the format {*m*}, {*m*,} or {*m*,*n*}, together with that interval expression it matches what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* will be decimal integers in the range $0 \le m \le n \le$ `{RE_DUP_MAX}`, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression {*m*} matches exactly *m* occurrences of the preceding ERE, {*m*,} matches at least *m* occurrences and {*m*,*n*} matches any number of occurrences between *m* and *n*, inclusive.

For example, in the string `abababccccccd` the ERE c{3} is matched by characters seven to nine and the ERE (ab){2,} is matched by characters one to six.

The behavior of multiple adjacent duplication symbols (`+`, `*`, `?` and intervals) produces undefined results.

ERE Alternation

Two EREs separated by the special character vertical-line (|) match a string that is matched by either. For example, the ERE a((bc)|d) matches the string abc and the string ad. Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, will be treated as an ERE matching a single character.

ERE Precedence

The order of precedence will be as shown in the following table:

ERE Precedence (from high to low)	
collation-related bracket symbols	[= =] [: :] [. .]
escaped characters	\<*special character*>
bracket expression	[ ]

grouping	( )
single-character-ERE duplication	* + ? {*m*,*n*}
concatenation	
anchoring	^ $
alternation	|

For example, the ERE `abba | cde` matches either the string `abba` or the string `cde` (rather than the string `abbade` or `abbcde`, because concatenation has a higher order of precedence than alternation).

ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters are considered ERE anchors when used anywhere outside a bracket expression. This has the following effects:

1. A circumflex (^) outside a bracket expression anchors the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs ^ab and (^ab) match ab in the string abcdef, but fail to match in the string cdefab, and the ERE a^b is valid, but can never match because the `a` prevents the expression ^b from matching starting at the first character.
2. A dollar sign (`$`) outside a bracket expression anchors the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs ef$ and (ef$) match ef in the string abcdef, but fail to match in the string cdefab, and the ERE e$f is valid, but can never match because the `f` prevents the expression e$ from matching ending at the last character.

See Also `localedef(1)`, `regcomp(3C)`, `attributes(5)`, `environ(5)`, `locale(5)`, `regexp(5)`

Name regexp, compile, step, advance – simple regular expression compile and match routines

Synopsis

```
#define INIT declarations
#define GETC(void) getc code
#define PEEKC(void) peekc code
#define UNGETC(void) ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

extern char *loc1, *loc2, *locs;

#include <regexp.h>

char *compile(char *instring, char *expbuf, const char *endfug, int eof);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);
```

Description Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings. The Simple Regular Expressions described below differ from the Internationalized Regular Expressions described on the regex(5) manual page in the following ways:

- only Basic Regular Expressions are supported
- the Internationalization features—character class, equivalence class, and multi-character collation—are not supported.

The functions `step()`, `advance()`, and `compile()` are general purpose regular expression matching routines to be used in programs that perform regular expression matching. These functions are defined by the `<regexp.h>` header.

The functions `step()` and `advance()` do pattern matching given a character string and a compiled regular expression as input.

The function `compile()` takes as input a regular expression as defined below and produces a compiled expression that can be used with `step()` or `advance()`.

Basic Regular Expressions A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. Some characters have special meaning when used in a regular expression; other characters stand for themselves.

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (`\`) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. `.`, `*`, `[`, and `\` (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets (`[ ]`; see 1.4 below).

b. `^` (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 4.1 and 4.3 below), or when it immediately follows the left of a pair of square brackets (`[ ]`) (see 1.4 below).

c. `$` (dollar sign), which is special at the `end` of an *entire* RE (see 4.2 below).

d. The character used to bound (that is, delimit) an entire RE, which is special for that RE (for example, see how slash (`/`) is used in the `g` command, below.)

1.3 A period (`.`) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets (`[ ]`) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (`^`), the one-character RE matches any character *except* new-line and the remaining characters in the string. The `^` has this special meaning *only* if it occurs first in the string. The minus (`–`) may be used to indicate a range of consecutive characters; for example, `[0–9]` is equivalent to `[0123456789]`. The `–` loses this special meaning if it occurs first (after an initial `^`, if any) or last in the string. The right square bracket (`]`) does not terminate such a string when it is the first character within it (after an initial `^`, if any); for example, `[ ]a–f]` matches either a right square bracket (`]`) or one of the ASCII letters `a` through `f` inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (`*`) is a RE that matches `0` or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by `\{`*m*`\}`, `\{`*m*`,\}`, or `\{`*m*`,`*n*`\}` is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; `\{`*m*`\}` matches *exactly m* occurrences; `\{`*m*`,\}` matches *at least m* occurrences; `\{`*m*`,`*n*`\}` matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences `\ (` and `\ )` is a RE that matches whatever the unadorned RE matches.

2.6 The expression `\` *n* matches the same string of characters as was matched by an expression enclosed between `\(` and `\)` *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of `\(` counting from the left. For example, the expression `^\(.*\)\1$` matches a line consisting of two repeated appearances of the same string.

An RE may be constrained to match words.

3.1 `\<` constrains a RE to match the beginning of a string or to follow a character that is not a digit, underscore, or letter. The first character matching the RE must be a digit, underscore, or letter.

3.2 `\>` constrains a RE to match the end of a string or to precede a character that is not a digit, underscore, or letter.

An *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

4.1 A circumflex (`^`) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

4.2 A dollar sign (`$`) at the end of an entire RE constrains that RE to match a *final* segment of a line.

4.3 The construction `^`*entire RE* `$` constrains the entire RE to match the entire line.

The null RE (for example, `//`) is equivalent to the last RE encountered.

Addressing with REs

Addresses are constructed as follows:

1. The character "`.`" addresses the current line.
2. The character "`$`" addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. *'x* addresses the line marked with the mark name character *x*, which must be an ASCII lower-case letter (`a`–`z`). Lines are marked with the `k` command described below.
5. A RE enclosed by slashes (`/`) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.
6. A RE enclosed in question marks (`?`) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line.

7. An address followed by a plus sign (+) or a minus sign (–) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. A shorthand for .+5 is .5.
8. If an address begins with + or –, the addition or subtraction is taken with respect to the current line; for example, –5 is understood to mean .–5.
9. If an address ends with + or –, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address – refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to –.) Moreover, trailing + and – characters have a cumulative effect, so —— refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair .,$.

Characters With Special Meaning

Characters that have special meaning except when they appear within square brackets ([ ]) or are preceded by \ are: ., *, [, \ . Other special characters, such as $ have special meaning in more restricted contexts.

The character ^ at the beginning of an expression permits a successful match only immediately after a newline, and the character $ at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character – denotes a range, [*c*–*c*], unless it is just after the open bracket or before the closing bracket, [–*c*] or [*c*–] in which case it has no special meaning. When used within brackets, the character ^ has the meaning *complement of* if it immediately follows the open bracket (example: [^*c*]); elsewhere between brackets (example: [*c*^]) it stands for the ordinary character ^.

The special meaning of the \ operator can be escaped only by preceding it with another \ , for example \\ .

Macros

Programs must have the following five macros declared before the `#include <regexp.h>` statement. These macros are used by the `compile()` routine. The macros `GETC`, `PEEKC`, and `UNGETC` operate on the regular expression given as input to `compile()`.

`GETC`	This macro returns the value of the next character (byte) in the regular expression pattern. Successive calls to `GETC` should return successive characters of the regular expression.
`PEEKC`	This macro returns the next character (byte) in the regular expression. Immediately successive calls to `PEEKC` should return the same character, which should also be the next character returned by `GETC`.
`UNGETC`	This macro causes the argument `c` to be returned by the next call to `GETC` and `PEEKC`. No more than one character of pushback is ever needed and this

character is guaranteed to be the last character read by GETC. The return value of the macro UNGETC(c) is always ignored.

RETURN(*ptr*) This macro is used on normal exit of the compile() routine. The value of the argument *ptr* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(*val*) This macro is the abnormal return from the compile() routine. The argument *val* is an error number (see ERRORS below for meanings). This call should never return.

compile() The syntax of the compile() routine is as follows:

compile(*instring*, *expbuf*, *endbuf*, *eof*)

The first parameter, *instring*, is never used explicitly by the compile() routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of (char *)0 for this parameter.

The next parameter, *expbuf*, is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (endbuf–expbuf) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. This character is usually a /.

Each program that includes the <regexp.h> header file must have a #define statement for INIT. It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for GETC, PEEKC, and UNGETC. Otherwise it can be used to declare external variables that might be used by GETC, PEEKC and UNGETC. (See EXAMPLES below.)

step(), advance() The first parameter to the step() and advance() functions is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter, *expbuf*, is the compiled regular expression which was obtained by a call to the function compile().

The function step() returns non-zero if some substring of *string* matches the regular expression in *expbuf* and 0 if there is no match. If there is a match, two external character pointers are set as a side effect to the call to step(). The variable loc1 points 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 input string, `loc1` will point to the first character of *string* and `loc2` will point to the null at the end of *string*.

The function `advance()` returns non-zero if the initial substring of *string* matches the regular expression in *expbuf*. If there is a match, an external character pointer, `loc2`, is set as a side effect. The variable `loc2` points to the next character in *string* after the last character that matched.

When `advance()` encounters a * or `\{ \}` sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, `advance()` will back up along the string until it finds a match or reaches the point in the string that initially matched the * or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer `locs` is equal to the point in the string at sometime during the backing up process, `advance()` will break out of the loop that backs up and will return zero.

The external variables `circf`, `sed`, and `nbra` are reserved.

Examples

EXAMPLE 1 Using Regular Expression Macros and Calls

The following is an example of how the regular expression macros and calls might be defined by an application program:

```
#define INIT       register char *sp = instring;
#define GETC()     (*sp++)
#define PEEKC()    (*sp)
#define UNGETC(c)  (——sp)
#define RETURN(c)  return;
#define ERROR(c)   regerr()

#include <regexp.h>
 . . .
     (void) compile(*argv, expbuf, &expbuf[ESIZE],'\0');
 . . .
     if (step(linebuf, expbuf))
                       succeed;
```

Diagnostics

The function `compile()` uses the macro `RETURN` on success and the macro `ERROR` on failure (see above). The functions `step()` and `advance()` return non-zero on a successful match and zero if there is no match. Errors are:

11 range endpoint too large.

16 bad number.

25 \ *digit* out of range.

36 illegal or missing delimiter.

41 no remembered search string.

42 \(\) imbalance.

43 too many \(.

44 more than 2 numbers given in \{ \}.

45 } expected after \.

46 first number exceeds second in \{ \}.

49 [ ] imbalance.

50 regular expression overflow.

See Also regex(5)

Name resource_controls – resource controls available through project database

Description The resource controls facility is configured through the project database. See `project(4)`. You can set and modify resource controls through the following utilities:

- `prctl(1)`
- `projadd(1M)`
- `projmod(1M)`
- `rctladm(1M)`

In a program, you use `setrctl(2)` to set resource control values.

In addition to the preceding resource controls, there are resource pools, accessible through the `pooladm(1M)` and `poolcfg(1M)` utilities. In a program, resource pools can be manipulated through the `libpool(3LIB)` library.

The following are the resource controls are available:

`process.max-address-space`
Maximum amount of address space, as summed over segment sizes, that is available to this process, expressed as a number of bytes.

`process.max-core-size`
Maximum size of a core file created by this process, expressed as a number of bytes.

`process.max-cpu-time`
Maximum CPU time that is available to this process, expressed as a number of seconds.

`process.max-data-size`
Maximum heap memory available to this process, expressed as a number of bytes.

`process.max-file-descriptor`
Maximum file descriptor index available to this process, expressed as an integer.

`process.max-file-size`
Maximum file offset available for writing by this process, expressed as a number of bytes.

`process.max-msg-messages`
Maximum number of messages on a message queue (value copied from the resource control at `msgget()` time), expressed as an integer.

`process.max-msg-qbytes`
Maximum number of bytes of messages on a message queue (value copied from the resource control at `msgget()` time), expressed as a number of bytes.

`process.max-port-events`
Maximum allowable number of events per event port, expressed as an integer.

`process.max-sem-nsems`
Maximum number of semaphores allowed per semaphore set, expressed as an integer.

`process.max-sem-ops`
Maximum number of semaphore operations allowed per `semop` call (value copied from the resource control at `semget()` time). Expressed as an integer, specifying the number of operations.

`process.max-stack-size`
Maximum stack memory segment available to this process, expressed as a number of bytes.

`project.cpu-caps`
Maximum amount of CPU resources that a project can use. The unit used is the percentage of a single CPU that can be used by all user threads in a project. Expressed as an integer. The cap does not apply to threads running in real-time scheduling class. This resource control does not support the `syslog` action.

`project.cpu-shares`
Number of CPU shares granted to a project for use with the fair share scheduler (see FSS(7)). The unit used is the number of shares (an integer). This resource control does not support the `syslog` action.

`project.max-contracts`
Maximum number of contracts allowed in a project, expressed as an integer.

`project.max-crypto-memory`
Maximum amount of kernel memory that can be used for crypto operations. Allocations in the kernel for buffers and session-related structures are charged against this resource control.

`project.max-locked-memory`
Total amount of physical memory locked by device drivers and user processes (including D/ISM), expressed as a number of bytes.

`project.max-lwps`
Maximum number of LWPs simultaneously available to a project, expressed as an integer.

`project.max-msg-ids`
Maximum number of message queue IDs allowed for a project, expressed as an integer.

`project.max-port-ids`
Maximum allowable number of event ports, expressed as an integer.

`project.max-sem-ids`
Maximum number of semaphore IDs allowed for a project, expressed as an integer.

`project.max-shm-ids`
Maximum number of shared memory IDs allowed for a project, expressed as an integer.

`project.max-shm-memory`
Total amount of shared memory allowed for a project, expressed as a number of bytes.

`project.max-tasks`
Maximum number of tasks allowable in a project, expressed as an integer.

`project.pool`
Binds a specified resource pool with a project.

`rcap.max-rss`
The total amount of physical memory, in bytes, that is available to processes in a project.

`task.max-cpu-time`
Maximum CPU time that is available to this task's processes, expressed as a number of seconds.

`task.max-lwps`
Maximum number of LWPs simultaneously available to this task's processes, expressed as an integer.

The following zone-wide resource controls are available:

`zone.cpu-cap`
Sets a limit on the amount of CPU time that can be used by a zone. The unit used is the percentage of a single CPU that can be used by all user threads in a zone. Expressed as an integer. When projects within the capped zone have their own caps, the minimum value takes precedence. This resource control does not support the `syslog` action.

`zone.cpu-shares`
Sets a limit on the number of fair share scheduler (FSS) CPU shares for a zone. CPU shares are first allocated to the zone, and then further subdivided among projects within the zone as specified in the `project.cpu-shares` entries. Expressed as an integer. This resource control does not support the `syslog` action.

`zone.max-locked-memory`
Total amount of physical locked memory available to a zone.

`zone.max-lwps`
Enhances resource isolation by preventing too many LWPs in one zone from affecting other zones. A zone's total LWPs can be further subdivided among projects within the zone within the zone by using `project.max-lwps` entries. Expressed as an integer.

`zone.max-msg-ids`
Maximum number of message queue IDs allowed for a zone, expressed as an integer.

`zone.max-sem-ids`
Maximum number of semaphore IDs allowed for a zone, expressed as an integer.

`zone.max-shm-ids`
Maximum number of shared memory IDs allowed for a zone, expressed as an integer.

`zone.max-shm-memory`
Total amount of shared memory allowed for a zone, expressed as a number of bytes.

`zone.max-swap`
Total amount of swap that can be consumed by user process address space mappings and `tmpfs` mounts for this zone.

See zones(5).

Units Used in Resource Controls

Resource controls can be expressed as in units of size (bytes), time (seconds), or as a count (integer). These units use the strings specified below.

Category	Res Ctrl Type String	Modifier	Scale
Size	bytes	B	1
		KB	2^10
		MB	2^20
		GB	2^30
		TB	2^40
		PB	2^50
		EB	2^60
Time	seconds	s	1
		Ks	10^3
		Ms	10^6
		Gs	10^9
		Ts	10^12
		Ps	10^15
		Es	10^18
Count	integer	none	1
		K	10^3
		M	10^6
		G	10^9
		T	10^12
		P	10^15
		Es	10^18

Scaled values can be used with resource controls. The following example shows a scaled threshold value:

```
task.max-lwps=(priv,1K,deny)
```

In the `project` file, the value `1K` is expanded to `1000`:

```
task.max-lwps=(priv,1000,deny)
```

A second example uses a larger scaled value:

```
process.max-file-size=(priv,5G,deny)
```

In the `project` file, the value `5G` is expanded to `5368709120`:

```
process.max-file-size=(priv,5368709120,deny)
```

The preceding examples use the scaling factors specified in the table above.

Note that unit modifiers (for example, `5G`) are accepted by the `prctl(1)`, `projadd(1M)`, and `projmod(1M)` commands. You cannot use unit modifiers in the project database itself.

Resource Control Values and Privilege Levels

A threshold value on a resource control constitutes a point at which local actions can be triggered or global actions, such as logging, can occur.

Each threshold value on a resource control must be associated with a privilege level. The privilege level must be one of the following three types:

`basic`
: Can be modified by the owner of the calling process.

`privileged`
: Can be modified by the current process (requiring `sys_resource` privilege) or by `prctl(1)` (requiring `proc_owner` privilege).

`system`
: Fixed for the duration of the operating system instance.

A resource control is guaranteed to have one `system` value, which is defined by the system, or resource provider. The `system` value represents how much of the resource the current implementation of the operating system is capable of providing.

Any number of privileged values can be defined, and only one basic value is allowed. Operations that are performed without specifying a privilege value are assigned a basic privilege by default.

The privilege level for a resource control value is defined in the privilege field of the resource control block as `RCTL_BASIC`, `RCTL_PRIVILEGED`, or `RCTL_SYSTEM`. See `setrctl(2)` for more information. You can use the `prctl` command to modify values that are associated with basic and privileged levels.

In specifying the privilege level of `privileged`, you can use the abbreviation `priv`. For example:

```
task.max-lwps=(priv,1K,deny)
```

Global and Local Actions on Resource Control Values

There are two categories of actions on resource control values: global and local.

Global actions apply to resource control values for every resource control on the system. You can use `rctladm(1M)` to perform the following actions:

- Display the global state of active system resource controls.
- Set global logging actions.

You can disable or enable the global logging action on resource controls. You can set the `syslog` action to a specific degree by assigning a severity level, `syslog=`*level*. The possible settings for *level* are as follows:

- `debug`

- info
- notice
- warning
- err
- crit
- alert
- emerg

By default, there is no global logging of resource control violations.

Local actions are taken on a process that attempts to exceed the control value. For each threshold value that is placed on a resource control, you can associate one or more actions. There are three types of local actions: none, deny, and signal=. These three actions are used as follows:

none
: No action is taken on resource requests for an amount that is greater than the threshold. This action is useful for monitoring resource usage without affecting the progress of applications. You can also enable a global message that displays when the resource control is exceeded, while, at the same time, the process exceeding the threshhold is not affected.

deny
: You can deny resource requests for an amount that is greater than the threshold. For example, a task.max-lwps resource control with action deny causes a fork() system call to fail if the new process would exceed the control value. See the fork(2).

signal=
: You can enable a global signal message action when the resource control is exceeded. A signal is sent to the process when the threshold value is exceeded. Additional signals are not sent if the process consumes additional resources. Available signals are listed below.

Not all of the actions can be applied to every resource control. For example, a process cannot exceed the number of CPU shares assigned to the project of which it is a member. Therefore, a deny action is not allowed on the project.cpu-shares resource control.

Due to implementation restrictions, the global properties of each control can restrict the range of available actions that can be set on the threshold value. (See rctladm(1M).) A list of available signal actions is presented in the following list. For additional information about signals, see signal(3HEAD).

The following are the signals available to resource control values:

SIGABRT
: Terminate the process.

SIGHUP
: Send a hangup signal. Occurs when carrier drops on an open line. Signal sent to the process group that controls the terminal.

`SIGTERM`
Terminate the process. Termination signal sent by software.

`SIGKILL`
Terminate the process and kill the program.

`SIGSTOP`
Stop the process. Job control signal.

`SIGXRES`
Resource control limit exceeded. Generated by resource control facility.

`SIGXFSZ`
Terminate the process. File size limit exceeded. Available only to resource controls with the `RCTL_GLOBAL_FILE_SIZE` property (`process.max-file-size`). See `rctlblk_set_value(3C)`.

`SIGXCPU`
Terminate the process. CPU time limit exceeded. Available only to resource controls with the `RCTL_GLOBAL_CPUTIME` property (`process.max-cpu-time`). See `rctlblk_set_value(3C)`.

Resource Control Flags and Properties

Each resource control on the system has a certain set of associated properties. This set of properties is defined as a set of flags, which are associated with all controlled instances of that resource. Global flags cannot be modified, but the flags can be retrieved by using either `rctladm(1M)` or the `setrctl(2)` system call.

Local flags define the default behavior and configuration for a specific threshold value of that resource control on a specific process or process collective. The local flags for one threshold value do not affect the behavior of other defined threshold values for the same resource control. However, the global flags affect the behavior for every value associated with a particular control. Local flags can be modified, within the constraints supplied by their corresponding global flags, by the `prctl` command or the `setrctl` system call. See `setrctl(2)`.

For the complete list of local flags, global flags, and their definitions, see `rctlblk_set_value(3C)`.

To determine system behavior when a threshold value for a particular resource control is reached, use `rctladm` to display the global flags for the resource control . For example, to display the values for `process.max-cpu-time`, enter:

```
$ rctladm process.max-cpu-time
process.max-cpu-time  syslog=off [ lowerable no-deny cpu-time inf seconds ]
```

The global flags indicate the following:

`lowerable`
Superuser privileges are not required to lower the privileged values for this control.

`no-deny`
Even when threshold values are exceeded, access to the resource is never denied.

`cpu-time`
`SIGXCPU` is available to be sent when threshold values of this resource are reached.

`seconds`
The time value for the resource control.

Use the `prctl` command to display local values and actions for the resource control. For example:

```
$ prctl -n process.max-cpu-time $$
    process 353939: -ksh
    NAME    PRIVILEGE    VALUE    FLAG   ACTION              RECIPIENT
 process.max-cpu-time
         privileged   18.4Es    inf   signal=XCPU                -
         system       18.4Es    inf   none
```

The `max` (`RCTL_LOCAL_MAXIMAL`) flag is set for both threshold values, and the `inf` (`RCTL_GLOBAL_INFINITE`) flag is defined for this resource control. An `inf` value has an infinite quantity. The value is never enforced. Hence, as configured, both threshold quantities represent infinite values that are never exceeded.

Resource Control Enforcement

More than one resource control can exist on a resource. A resource control can exist at each containment level in the process model. If resource controls are active on the same resource at different container levels, the smallest container's control is enforced first. Thus, action is taken on `process.max-cpu-time` before `task.max-cpu-time` if both controls are encountered simultaneously.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also

`prctl(1)`, `pooladm(1M)`, `poolcfg(1M)`, `projadd(1M)`, `projmod(1M)`, `rctladm(1M)`, `setrctl(2)`, `rctlblk_set_value(3C)`, `libpool(3LIB)`, `project(4)`, `attributes(5)`, `FSS(7)`

System Administration Guide: Virtualization Using the Solaris Operating System

Name sgml, solbook – Standard Generalized Markup Language

Description Standard Generalized Markup Language (SGML) is the ISO standard 8879:1986 that describes a syntax for marking up documents with tags that describe the purpose of the text rather than the appearance on the page. This form of markup facilitates document interchange between different platforms and applications. SGML allows the management of information as data objects rather than text on a page.

In an SGML document the main structural components are called `elements`. The organization and structure of a document and the meaning of elements are described in the Document Type Definition (DTD). Elements are the *tags* that identify the content. Element names may be descriptive of the content for ease of use. For example `<para>` for paragraphs. Elements can have `attributes` which are used to modify or refine the properties or characteristics of the element. Within the DTD a valid context for each element is defined and a framework is provided for the types of elements that constitute a compliant document.

Another component of the DTD is `entities`. Entities are a collection of characters that can be referenced as a unit. Entities are similar to constants in a programming language such as C. They can be defined and referenced. An entity can represent one character or symbol which does not appear on a standard keyboard, a word or group of words, or an entire separate sgml marked-up file. Entities allow reuse of standard text.

There is no single standard DTD , but the de facto standard for the computer industry is the DocBook DTD , developed and maintained by the Davenport Group. Within Sun, the SolBook DTD , which is a proper subset of DocBook DTD , is used when writing reference manual pages. The SolBook DTD contains a number of tags that are designed for the unique needs of the reference pages.

SolBook Elements Elements are defined with a hierarchical structure that gives a structure to the document. The following is a description of some of the elements from the SolBook DTD which are used for reference pages.

DOCTYPE The first line in an SGML file that identifies the location of the DTD that is used to define the document. The `<!DOCTYPE` string is what the SGML -aware `man(1)` command uses to identify that a file is formatted in SGML rather than `nroff(1)`.

RefEntry The top layer element that contains a reference page is `<refentry>`. All of the text and other tags must be contained within this tag.

RefMeta The next tag in a reference page is `<refmeta>`, which is a container for several other tags. They are:

<refentrytitle> This is the title of the reference page. It is equivalent to the name of the reference page's file name, without the section number extension.

<manvolnum> This is the section number that the reference page resides in. The contents may be a text entity reference.

<refmiscinfo> There are one or more <refmiscinfo> tags which contain *meta* information. Meta information is information about the reference page. The <refmiscinfo> tag has the class attribute. There are four classes that are routinely used.

- date — This is the date that the file was last modified. By consensus this date is changed only when the technical information on the page changes and not simply for an editorial change.
- sectdesc — This is the section title of the reference page; for example User Commands. The value of this attribute may be a text entity reference.
- software — This is the name of the software product that the topic discussed on the reference page belongs to. For example UNIX commands are part of the SunOS x.x release. The value of this attribute may be a text entity reference.
- arch — This is the architectural platform limitation of the subject discussed on the reference page. If there are no limitations the value used is generic. Other values are sparc and x86.
- copyright — This attribute contains the Sun Microsystems copyright. Any other copyrights that may pertain to the individual reference page file should be entered as separate <refmiscinfo> entries. The value of this attribute may be a text entity reference.

RefNameDiv This tag contains the equivalent information to the .TH macro line in an nroff(1) reference page. <refnamediv> contains three tags. These tags contain the text that is before and after the '–' (dash) on the NAME line.

<refname> These are the names of the topics that are discussed in the file. There may be more than one <refname> for a page. The first <refname> must match the name of the file and the <refentrytitle>. If there are more than one <refname> tags, each is separated by a ',' (comma). The comma is generated by the publisher of sgml files, so it should not be typed. This is referred to as *auto-generated* text.

<refpurpose> The text after the dash on the NAME line is contained in this tag. This is a short summary of what the object or objects described on the reference page do or are used for. The dash is also auto-generated and should not be typed in.

<refdiscriptor> In some cases the <refentrytitle> is a general topic descriptor of a group of related objects that are discussed on the same page. In this case the first tag after the <refnamediv> is a <refdiscriptor>. The <refname> tags follow. Only one <refdiscriptor> is allowed, and it

should match the <refentrytitle>.

RefSynopsisDiv The SYNOPSIS line of the reference page is contained by this tag. There is a <title> that usually contains an entity reference. The text is the word SYNOPSIS. There are several tags within <refsynopsisdiv> that are designed specifically for the type of synopsis that is used in the different reference page sections. The three types are:

<cmdsynopsis>	Used for commands and utilities pages.
<funcsynopsis>	Used for programming interface pages.
<synopsis>	Used for pages that do not fall into the other two categories.

RefSect1 This tag is equivalent to the .SH nroff macro. It contains a <title> element that is the title of the reference page section. Section names are the standard names such as DESCRIPTION, OPTIONS, PARAMETERS, SEE ALSO, and others. The contents of the <title> may be a text entity reference.

RefSect2 This tag is equivalent to the .SS nroff macro. It contains a <title> element that contains the text of the sub-section heading. <refsect2> tags may also be used within a <refsynopsisdiv> as a sub-section heading for the SYNOPSIS section.

Block Elements There are a number of block elements that are used for grouping text. This is a list of some of these elements.

<para>	This tag is used to contain a paragraph of text.
<variablelist>	This tag is used to create two column lists. For example descriptions for command options, where the first column lists the option and the second column describes the option.
<orderedlist>	An list of items in a specific order.
<itemizedlist>	A list of items that are marked with a character such as a bullet or a dash.
<literallayout>	Formatted program output as produced by a program or command. This tag is a container for lines set off from the main text in which line breaks, tabs, and leading white space are significant.
<programlisting>	A segment of program code. Line breaks and leading white space are significant.
<table>	This tag contains the layout and content for tabular formatting of information. <table> has a required <title>.
<informaltable>	This tag is the same as the <table> tag except the <title> is not required.

<example>	This tag contains examples of source code or usage of commands. It contains a required `<title>`.
<informalexample>	This tag is the same as the `<example>` tag except the `<title>` is not required.

Inline Elements The inline elements are used for tagging text.

<command>	An executable program or the entry a user makes to execute a command.
<function>	A subroutine in a program or external library.
<literal>	Contains any literal string.
<parameter>	An argument passed to a computer program by a function or routine.
<inlineequation>	An untitled mathematical equation occurring in-line.
<link>	A hypertext link to text within a book, in the case of the reference manual it is used to cross reference to another reference page.
<olink>	A hypertext link used to create cross references to books other than the reference manual.
<xref>	A cross reference to another part of the same reference page.

See Also `man(1)`, `nroff(1)`, `man(5)`

Name smf – service management facility

Description The Solaris service management facility defines a programming model for providing persistently running applications called *services*. The facility also provides the infrastructure in which to run services. A service can represent a running application, the software state of a device, or a set of other services. Services are represented in the framework by *service instance* objects, which are children of service objects. Instance objects can inherit or override the configuration of the parent service object, which allows multiple service instances to share configuration information. All service and instance objects are contained in a *scope* that represents a collection of configuration information. The configuration of the local Solaris instance is called the "localhost" scope, and is the only currently supported scope.

Each service instance is named with a fault management resource identifier (FMRI) with the scheme `svc:`. For example, the `syslogd(1M)` daemon started at system startup is the default service instance named:

```
svc://localhost/system/system-log:default
svc:/system/system-log:default
system/system-log:default
```

Many commands also allow FMRI abbreviations. See the `svcs(1)` man page for one such example.

In the above example, 'default' is the name of the instance and 'system/system-log' is the service name. Service names can comprise multiple components separated by slashes (/). All components, except the last, compose the *category* of the service. Site-specific services should be named with a category beginning with 'site'.

A service instance is either enabled or disabled. All services can be enabled or disabled with the `svcadm(1M)` command.

The list of managed service instances on a system can be displayed with the `svcs(1)` command.

Dependencies Service instances can have dependencies on a set of `entities` which can include services and files. Dependencies govern when the service is started and automatically stopped. When the dependencies of an enabled service are not satisfied, the service is kept in the offline state. When its dependencies are satisfied, the service is started. If the start is successful, the service is transitioned to the online state.

Whether a dependency is satisfied is determined by its `grouping`:

`require_all`	Satisfied when all cited services are running (online or degraded), or when all indicated files are present.
`require_any`	Satisfied when one of the cited services is running (online or degraded), or when at least one of the indicated files is present.

`optional_all` Satisfied if the cited services are running (online or degraded) or do not run without administrative action (disabled, maintenance, not present, or offline waiting for dependencies which do not start without administrative action).

`exclude_all` Satisfied when all of the cited services are disabled, in the maintenance state, or when cited services or files are not present.

Once running (online or degraded), if a service cited by a `require_all`, `require_any`, or `optional_all` dependency is stopped or refreshed, the SMF considers why the service was stopped and the `restart_on` attribute of the dependency to decide whether to stop the service.

```
                   |  restart_on value
event              |  none  error restart refresh
-------------------+-----------------------------
stop due to error  |  no    yes   yes     yes
non-error stop     |  no    no    yes     yes
refresh            |  no    no    no      yes
```

A service is considered to have stopped due to an error if the service has encountered a hardware error or a software error such as a core dump. For `exclude_all` dependencies, the service is stopped if the cited service is started and the `restart_on` attribute is not `none`.

The dependencies on a service can be listed with `svcs(1)` or `svccfg(1M)`, and modified with `svccfg(1M)`.

Restarters Each service is managed by a restarter. The master restarter, `svc.startd(1M)` manages states for the entire set of service instances and their dependencies. The master restarter acts on behalf of its services and on delegated restarters that can provide specific execution environments for certain application classes. For instance, `inetd(1M)` is a delegated restarter that provides its service instances with an initial environment composed of a network connection as input and output file descriptors. Each instance delegated to `inetd(1M)` is in the online state. While the daemon of a particular instance might not be running, the instance is available to run.

As dependencies are satisfied when instances move to the online state, `svc.startd(1M)` invokes start methods of other instances or directs the delegated restarter to do so. These operations might overlap.

The current set of services and associated restarters can be examined using `svcs(1)`. A description of the common configuration used by all restarters is given in `smf_restarter(5)`.

Methods Each service or service instance must define a set of methods that start, stop, and, optionally, refresh the service. See `smf_method(5)` for a more complete description of the method conventions for `svc.startd(1M)` and similar `fork(2)`-`exec(2)` restarters.

Administrative methods, such as for the capture of legacy configuration information into the repository, are discussed on the svccfg(1M) manual page.

The methods for a service can be listed and modified using the svccfg(1M) command.

States Each service instance is always in a well-defined state based on its dependencies, the results of the execution of its methods, and its potential contracts events. The following states are defined:

UNINITIALIZED	This is the initial state for all service instances. Instances are moved to maintenance, offline, or a disabled state upon evaluation by svc.startd(1M) or the appropriate restarter.
OFFLINE	The instance is enabled, but not yet running or available to run. If restarter execution of the service start method or the equivalent method is successful, the instance moves to the online state. Failures might lead to a degraded or maintenance state. Administrative action can lead to the uninitialized state.
ONLINE	The instance is enabled and running or is available to run. The specific nature of the online state is application-model specific and is defined by the restarter responsible for the service instance. Online is the expected operating state for a properly configured service with all dependencies satisfied. Failures of the instance can lead to a degraded or maintenance state. Failures of services on which the instance depends can lead to offline or degraded states.
DEGRADED	The instance is enabled and running or available to run. The instance, however, is functioning at a limited capacity in comparison to normal operation. Failures of the instance can lead to the maintenance state. Failures of services on which the instance depends can lead to offline or degraded states. Restoration of capacity should result in a transition to the online state.
MAINTENANCE	The instance is enabled, but not able to run. Administrative action (through svcadm clear) is required to move the instance out of the maintenance state. The maintenance state might be a temporarily reached state if an administrative operation is underway.
DISABLED	The instance is disabled. Enabling the service results in a transition to the offline state and eventually to the online state with all dependencies satisfied.
LEGACY-RUN	This state represents a legacy instance that is not managed by the service management facility. Instances in this state have been started at some point, but might or might not be running. Instances can only be observed using the facility and are not transferred into other states.

States can also have transitions that result in a return to the originating state.

Properties and Property Groups

The dependencies, methods, delegated restarter, and instance state mentioned above are represented as properties or property groups of the service or service instance. A service or service instance has an arbitrary number of property groups in which to store application data. Using property groups in this way allows the configuration of the application to derive the attributes that the repository provides for all data in the facility. The application can also use the appropriate subset of the `service_bundle(4)` DTD to represent its configuration data within the framework.

Property lookups are composed. If a property group-property combination is not found on the service instance, most commands and the high-level interfaces of `libscf(3LIB)` search for the same property group-property combination on the service that contains that instance. This allows common configuration among service instances to be shared. Composition can be viewed as an inheritance relationship between the service instance and its parent service.

Properties are protected from modification by unauthorized processes. See `smf_security(5)`.

General Property Group

The `general` property group applies to all service instances. It includes the following properties:

enabled (boolean) — Specifies whether the instance is enabled. If this property is not present on an instance, SMF does not tell the instance's restarter about the existence of the restarter.

restarter (fmri) — The restarter for this service. See the Restarters section for more information. If this property is unset, the default system restarter is used.

Snapshots

Historical data about each instance in the repository is maintained by the service management facility. This data is made available as read-only snapshots for administrative inspection and rollback. The following set of snapshot types might be available:

`initial` — Initial configuration of the instance created by the administrator or produced during package installation.

`last_import` — Configuration as prescribed by the manifest of the service that is taken during `svccfg(1M)` import operation. This snapshot provides a baseline for determining property customization.

`previous` — Current configuration captured when an administrative undo operation is performed.

`running` — The running configuration of the instance.

`start` — Configuration captured during a successful transition to the online state.

The svccfg(1M) command can be used to interact with snapshots.

Special Property Groups

Some property groups are marked as "non-persistent". These groups are not backed up in snapshots and their content is cleared during system boot. Such groups generally hold an active program state which does not need to survive system restart.

Configuration Repository

The current state of each service instance, as well as the properties associated with services and service instances, is stored in a system repository managed by svc.configd(1M). This repository is transactional and able to provide previous versions of properties and property groups associated with each service or service instance.

The repository for service management facility data is managed by svc.configd(1M).

Service Bundles, Manifests, and Profiles

The information associated with a service or service instance that is stored in the configuration repository can be exported as XML-based files. Such XML files, known as service bundles, are portable and suitable for backup purposes. Service bundles are classified as one of the following types:

`manifests` Files that contain the complete set of properties associated with a specific set of services or service instances.

`profiles` Files that contain a set of service instances and values for the enabled property (type `boolean` in the general property group) on each instance.

Profiles can also contain configuration values for properties in services and instances. Template elements cannot be defined in a profile.

Service bundles can be imported or exported from a repository using the svccfg(1M) command. See service_bundle(4) for a description of the service bundle file format with guidelines for authoring service bundles.

A *service archive* is an XML file that contains the description and persistent properties of every service in the repository, excluding transient properties such as service state. This service archive is basically a 'svccfg export' for every service which is not limited to named services.

Milestones

An `smf` milestone is a service that aggregates a multiple service dependencies. Usually, a milestone does nothing useful itself, but declares a specific state of system-readiness on which other services can depend. One example is the `name-services` milestone, which simply depends upon the currently enabled name services.

Legacy Startup Scripts

Startup programs in the `/etc/rc?.d` directories are executed as part of the corresponding run-level milestone:

`/etc/rcS.d` milestone/single-user:default

`/etc/rc2.d` milestone/multi-user:default

`/etc/rc3.d` milestone/multi-user-server:default

Execution of each program is represented as a reduced-functionality service instance named by the program's path. These instances are held in a special legacy-run state.

These instances do not have an enabled property (type `boolean` in the general property group) and, generally, cannot be manipulated with the `svcadm`(1M) command. No error diagnosis or restart is done for these programs.

See Also `svcs`(1), `inetd`(1M), `svcadm`(1M), `svccfg`(1M), `svc.configd`(1M), `svc.startd`(1M), `exec`(2), `fork`(2), `libscf`(3LIB), `strftime`(3C), `contract`(4), `service_bundle`(4), `smf_bootstrap`(5), `smf_method`(5), `smf_restarter`(5), `smf_security`(5)

Name smf_bootstrap – service management facility boot, packaging, and compatibility behavior

Description The service management facility establishes conventions for delivering service manifests, incorporating service manifest changes, describing service configuration stability, using service configuration overrides, and the use of service profiles.

Manifest Loading at Boot

The `svc:/system/manifest-import:default` service uses `svccfg(1M)` to import certain manifest files from the `/var/svc/manifest` directory tree into the service configuration repository. The service imports files that it has not imported previously and those files which have changed since the last time they were imported by the service. When a manifest is imported by the service, a hash of the file that includes its contents is recorded in a property group of the `svc:/smf/manifest` service. The `manifest-import` service uses the hash to determine whether the file has changed. See `svccfg(1M)` for information on the `svccfg import` behavior for services that already exist in the repository.

Manifest Handling During Packaging Operations

Service manifests within packages should be identified with the class `manifest`. Class action scripts that install and remove service manifests are included in the packaging subsystem. When `pkgadd(1M)` is invoked, the service manifest is imported.

When `pkgrm(1M)` is invoked, instances in the manifest that are disabled are deleted. Instances in the manifest that are online or degraded are disabled first and then deleted. Any services in the manifest with no remaining instances are also deleted.

If the `-R` option is supplied to `pkgadd(1M)` or `pkgrm(1M)`, the actions described in this section will be done when the system is next rebooted with that alternate root path.

Stability Declarations

Each service group and each property group delivered in a manifest should declare a stability level based on `attributes(5)` definitions. With knowledge of the stability level, an application developer can determine the likelihood that feature development based on the existence or components of a service or object is likely to remain functional across a release boundary.

In an `smf(5)` context, the stability value also identifies the expected scope of the changes to properties within the property group across a release boundary for the service, which can include patches for that service. The following two sections discuss this in more detail.

Property Overrides

The `service_bundle(4)` document type definition includes an override attribute that is applicable to each property in a service manifest. If set to `true`, the attribute instructs `svccfg(1M)` and other manifest import tools to replace the current value of a property in the repository with the one from the manifest. If the override attribute is absent or present but set to `false`, the current value in the repository is preserved.

Property groups declared as Stable do not contain override attributes on enclosed properties. Property groups declared as Evolving do so only to correct an erroneous setting. Property groups declared as Unstable can contain overrides on any property. The exception to this behavior is for the stability property itself, which can be modified to identify incipient change to the interface presented by the service.

Property Group Deletion

The service_bundle(4) document type definition includes a delete attribute, applicable to each property group in a service manifest. If set to `true`, the delete attribute instructs svccfg(1M) and other manifest import tools to delete this property group from the repository. If the delete attribute is absent or present but set to `false`, the property group in the repository is preserved.

Property groups declared as Stable or Evolving are not deleted. Property groups declared as Unstable can be deleted across any release boundary.

Profile Application

The first time the existence of each of the three service profiles listed below is detected, svc.startd(1M) automatically applies the profile.

```
/var/svc/profile/generic.xml
/var/svc/profile/platform.xml
/var/svc/profile/site.xml
```

The `svc:/smf/manifest` service is used in a similar fashion.

Additional service profiles that characterize the activation of various groups of service instances might be present in `/var/svc/profile`. None of the `/var/svc/profile` profiles are automatically applied to the repository. A profile can be manually applied or re-applied using svccfg(1M).

See Also pkgadd(1M), pkgrm(1M), svcadm(1M), svccfg(1M), svc.startd(1M), libscf(3LIB), service_bundle(4), attributes(5), smf(5), smf_security(5)

Notes The present version of smf(5) does not support multiple repositories.

Name smf_method – service management framework conventions for methods

Description The class of services managed by svc.startd(1M) in the service management framework, smf(5), consists of applications that fit a simple fork(2)-exec(2) model. The svc.startd(1M) master daemon and other restarters support the fork(2)-exec(2) model, potentially with additional capabilities. The svc.startd(1M) daemon and other restarters require that the methods which activate, manipulate, or examine a service instance follow the conventions described in this manual page.

Invocation form The form of a method invocation is not dictated by convention. In some cases, a method invocation might consist of the direct invocation of the daemon or other binary executable that provides the service. For cases in which an executable script or other mediating executable is used, the convention recommends the form:

```
/path/to/method_executable abbr_method_name
```

The *abbr_method_name* used for the recommended form is a supported method such as `start` or `stop`. The set of methods supported by a restarter is given on the related restarter page. The svc.startd(1M) daemon supports `start`, `stop`, and `refresh` methods.

A restarter might define other kinds of methods beyond those referenced in this page. The conventions surrounding such extensions are defined by the restarter and might not be identical to those given here.

Environment Variables The restarter provides four environment variables to the method that determine the context in which the method is invoked.

`SMF_FMRI`
: The service fault management resource identifier (FMRI) of the instance for which the method is invoked.

`SMF_METHOD`
: The full name of the method being invoked, such as `start` or `stop`.

`SMF_RESTARTER`
: The service FMRI of the restarter that invokes the method

`SMF_ZONENAME`
: The name of the zone in which the method is running. This can also be obtained by using the zonename(1) command.

These variables should be removed from the environment prior to the invocation of any persistent process by the method. A convenience shell function, `smf_clear_env`, is given for service authors who use Bourne-compatible shell scripting to compose service methods in the include file described below.

The method context can cause other environment variables to be set as described below.

Method Definition A method is defined minimally by three properties in a propertygroup of type `method`.

These properties are:

exec (*astring*)	Method executable string.
timeout_seconds (*count*)	Number of seconds before method times out. See the `Timeouts` section for more detail.
type (*astring*)	Method type. Currently always set to `method`.

A Method Context can be defined to further refine the execution environment of the method. See the `Method Context` section for more information.

Method Tokens When defined in the `exec` string of the method by the restarter `svc.startd`, a set of tokens are parsed and expanded with appropriate value. Other restarters might not support method tokens. The delegated restarter for inet services, `inetd(1M)`, does not support the following method expansions.

`%%`
: %

`%r`
: Name of the restarter, such as `svc.startd`

`%m`
: The full name of the method being invoked, such as `start` or `stop`.

`%s`
: Name of the service

`%i`
: Name of the instance

`%f`
: FMRI of the instance

`%{prop[:,]}`
: Value(s) of a property. The `prop` might be a property FMRI, a property group name and a property name separated by a `/`, or a property name in the `application` property group. These values can be followed by a `,` (comma) or `:` (colon). If present, the separators are used to separate multiple values. If absent, a space is used. The following shell metacharacters encountered in string values are quoted with a \ (backslash):

  ```
  ; & ( ) | ^ < > newline space tab  \  " '
  ```

  An invalid expansion constitutes method failure.

Two explicit tokens can be used in the place of method commands.

`:kill [-signal]`
Sends the specified signal, which is `SIGTERM` by default, to all processes in the primary instance contract. Always returns `SMF_EXIT_OK`. This token should be used to replace common `pkill` invocations.

`:true`
Always returns `SMF_EXIT_OK`. This token should be used for methods that are required by the restarter but which are unnecessary for the particular service implementation.

Exiting and Exit Status

The required behavior of a start method is to delay exiting until the service instance is ready to answer requests or is otherwise functional.

The following exit status codes are defined in `<libscf.h>` and in the shell support file.

`SMF_EXIT_OK`	`0`	Method exited, performing its operation successfully.
`SMF_EXIT_ERR_FATAL`	`95`	Method failed fatally and is unrecoverable without administrative intervention.
`SMF_EXIT_ERR_CONFIG`	`96`	Unrecoverable configuration error. A common condition that returns this exit status is the absence of required configuration files for an enabled service instance.
`SMF_EXIT_ERR_NOSMF`	`99`	Method has been mistakenly invoked outside the `smf(5)` facility. Services that depend on `smf(5)` capabilities should exit with this status value.
`SMF_EXIT_ERR_PERM`	`100`	Method requires a form of permission such as file access, privilege, authorization, or other credential that is not available when invoked.
`SMF_EXIT_ERR_OTHER`	`non-zero`	Any non-zero exit status from a method is treated as an unknown error. A series of unknown errors can be diagnosed as a fault by the restarter or on behalf of the restarter.

Use of a precise exit code allows the responsible restarter to categorize an error response as likely to be intermittent and worth pursuing restart or permanent and request administrative intervention.

Timeouts

Each method can have an independent timeout, given in seconds. The choice of a particular timeout should be based on site expectations for detecting a method failure due to non-responsiveness. Sites with replicated filesystems or other failover resources can elect to lengthen method timeouts from the default. Sites with no remote resources can elect to shorten the timeouts. Method timeout is specified by the `timeout_seconds` property.

If you specify `0 timeout_seconds` for a method, it declares to the restarter that there is no timeout for the service. This setting is not preferred, but is available for services that absolutely require it.

`-1 timeout_seconds` is also accepted, but is a deprecated specification.

Shell Programming Support

A set of environment variables that define the above exit status values is provided with convenience shell functions in the file `/lib/svc/share/smf_include.sh`. This file is a Bourne shell script suitable for inclusion via the source operator in any Bourne-compatible shell.

To assist in the composition of scripts that can serve as SMF methods as well as `/etc/init.d` scripts, the `smf_present()` shell function is provided. If the `smf(5)` facility is not available, `smf_present()` returns a non-zero exit status.

One possible structure for such a script follows:

```
if smf_present; then
      # Shell code to run application as managed service
      ....

      smf_clear_env
else
      # Shell code to run application as /etc/init.d script
      ....
fi
```

This example shows the use of both convenience functions that are provided.

Method Context

The service management facility offers a common mechanism set the context in which the `fork(2)`-`exec(2)` model services execute.

The desired method context should be provided by the service developer. All service instances should run with the lowest level of privileges possible to limit potential security compromises.

A method context can contain the following properties:

`use_profile`
: A boolean that specifies whether the profile should be used instead of the `user`, `group`, `privileges`, and `limit_privileges` properties.

environment
: Environment variables to insert into the environment of the method, in the form of a number of `NAME=value` strings.

`profile`
: The name of an RBAC (role-based access control) profile which, along with the method executable, identifies an entry in `exec_attr(4)`.

`user`
: The user ID in numeric or text form.

`group`
The group ID in numeric or text form.

`supp_groups`
An optional string that specifies the supplemental group memberships by ID, in numeric or text form.

`privileges`
An optional string specifying the privilege set as defined in `privileges(5)`.

`limit_privileges`
An optional string specifying the limit privilege set as defined in `privileges(5)`.

`working_directory`
The home directory from which to launch the method. `:home` can be used as a token to indicate the home directory of the user whose `uid` is used to launch the method. If the property is unset, `:home` is used.

`corefile_pattern`
An optional string that specifies the corefile pattern to use for the service, as per `coreadm(1M)`. Most restarters supply a default. Setting this property overrides local customizations to the global core pattern.

`project`
The project ID in numeric or text form. `:default` can be used as a token to indicate a project identified by `getdefaultproj(3PROJECT)` for the user whose `uid` is used to launch the method.

`resource_pool`
The resource pool name on which to launch the method. `:default` can be used as a token to indicate the pool specified in the `project(4)` entry given in the `project` attribute above.

The method context can be set for the entire service instance by specifying a `method_context` property group for the service or instance. A method might override the instance method context by providing the method context properties on the method property group.

Invalid method context settings always lead to failure of the method, with the exception of invalid environment variables that issue warnings.

In addition to the context defined above, many `fork(2)-exec(2)` model restarters also use the following conventions when invoking executables as methods:

Argument array
The arguments in `argv[]` are set consistently with the result `/bin/sh -c` of the `exec` string.

File descriptors
File descriptor `0` is `/dev/null`. File descriptors `1` and `2` are recommended to be a per-service log file.

Files `/lib/svc/share/smf_include.sh`
Definitions of exit status values.

`/usr/include/libscf.h`
Definitions of exit status codes.

See Also `zonename(1)`, `coreadm(1M)`, `inetd(1M)`, `svccfg(1M)`, `svc.startd(1M)`, `exec(2)`, `fork(2)`, `getdefaultproj(3PROJECT)`, `exec_attr(4)`, `project(4)`, `service_bundle(4)`, `attributes(5)`, `privileges(5)`, `rbac(5)`, `smf(5)`, `smf_bootstrap(5)`, `zones(5)`

Notes The present version of `smf(5)` does not support multiple repositories.

When a service is configured to be started as root but with privileges different from `limit_privileges`, the resulting process is privilege aware. This can be surprising to developers who expect `seteuid(<non-zero UID>)` to reduce privileges to basic or less.

Name smf_restarter – service management facility conventions for restarters

Description All service instances in the service management facility must be managed by a restarter. This manual page describes configuration, functionality, and reporting characteristics that are common to all restarters in the framework. Characteristics specific to a particular restarter are described in the restarter's man page.

For each managed service, a restarter relies on retrieving properties on the service instance to determine configuration. The restarter manages a set of property groups to communicate the current disposition of a service with display tools such as `svcs(1)`.

Service Configuration The common restarter configuration for all services is captured in the `general` property group. This group includes the following required and optional property settings.

`enabled`	This is a required property. If set, the restarter of an instance attempts to maintain availability of the service.
`restarter`	This is an optional property that allows the specification of an alternate restarter to manage the service instance. If the restarter property is empty or absent, the restarter defaults to `svc.startd(1M)`.
`single_instance`	This is an optional property. When set, only one instance of the service is allowed to transition to an online or degraded status at any time.

Service Reporting All restarters report status using the `restarter` property group, which includes the following properties:

`next_state`	The current state and next state, if currently in transition, for instances stored in these properties. See `smf(5)` for a description of the potential states.
`auxiliary_state`	An astring with no spaces that contains a precise term to describe the full restarter-specific state in combination with the restarter state property. The auxiliary state cannot always be set and is always cleared during transition out of any state. Each restarter must define the precise list of auxiliary states it uses.
`state_timestamp`	The time when the current state was reached.
`contract`	The primary process contract ID, if any, under which the service instance is executing.

See Also `svcs(1)`, `svc.startd(1M)`, `service_bundle(4)`, `smf(5)`, `smf_method(5)`

Name smf_security – service management facility security behavior

Description The configuration subsystem for the service management facility, smf(5), requires privilege to modify the configuration of a service. Privileges are granted to a user by associating the authorizations described below to the user through user_attr(4) and prof_attr(4). See rbac(5).

The following authorization is used to manipulate services and service instances.

`solaris.smf.modify`	Authorized to add, delete, or modify services, service instances, or their properties, and to read protected property values.

Property Group Authorizations

The smf(5) configuration subsystem associates properties with each service and service instance. Related properties are grouped. Groups can represent an execution method, credential information, application data, or restarter state. The ability to create or modify property groups can cause smf(5) components to perform actions that can require operating system privilege. Accordingly, the framework requires appropriate authorization to manipulate property groups.

Each property group has a type corresponding to its purpose. The core property group types are `method`, `dependency`, `application`, and `framework`. Additional property group types can be introduced, provided they conform to the extended naming convention in smf(5). The following basic authorizations, however, apply only to the core property group types:

`solaris.smf.modify.method`	Authorized to change values or create, delete, or modify a property group of type `method`.
`solaris.smf.modify.dependency`	Authorized to change values or create, delete, or modify a property group of type `dependency`.
`solaris.smf.modify.application`	Authorized to change values, read protected values, and create, delete, or modify a property group of type application.
`solaris.smf.modify.framework`	Authorized to change values or create, delete, or modify a property group of type `framework`.
`solaris.smf.modify`	Authorized to add, delete, or modify services, service instances, or their properties, and to read protected property values.

Property group-specific authorization can be specified by properties contained in the property group.

`modify_authorization`	Authorizations allow the addition, deletion, or modification of properties within the property group, and the retrieval of property values from the property group if protected.

`value_authorization` — Authorizations allow changing the values of any property of the property group except `modify_authorization`, and the retrieval of any property values except modify_authorization from the property group if protected.

`read_authorization` — Authorizations allow the retrieval of property values within the property group. The presence of a string-valued property with this name identifies the containing property group as protected. This property has no effect on property groups of types other than application. See `Protected Property Groups`.

The above authorization properties are only used if they have type `astring`. If an instance property group does not have one of the properties, but the instance's service has a property group of the same name with the property, its values are used.

Protected Property Groups

Normally, all property values in the repository can be read by any user without explicit authorization. Property groups of non-framework types can be used to store properties with values that require protection. They must not be revealed except upon proper authorization. A property group's status as protected is indicated by the presence of a string-valued `read_authorization` property. If this property is present, the values of all properties in the property group is retrievable only as described in `Property Group Authorizations`.

Administrative domains with policies that prohibit backup of data considered sensitive should exclude the SMF repository databases from their backups. In the face of such a policy, non-protected property values can be backed up by using the `svccfg(1M)` archive command to create an archive of the repository without protected property values.

Service Action Authorization

Certain actions on service instances can result in service interruption or deactivation. These actions require an authorization to ensure that any denial of service is a deliberate administrative action. Such actions include a request for execution of the refresh or restart methods, or placement of a service instance in the maintenance or other non-operational state. The following authorization allows such actions to be requested:

`solaris.smf.manage` — Authorized to request restart, refresh, or other state modification of any service instance.

In addition, the `general/action_authorization` property can specify additional authorizations that permit service actions to be requested for that service instance. The `solaris.smf.manage` authorization is required to modify this property.

Defined Rights Profiles

Two rights profiles are included that offer grouped authorizations for manipulating typical `smf(5)` operations.

Service Management — A service manager can manipulate any service in the repository in any way. It corresponds to the `solaris.smf.manage` and `solaris.smf.modify` authorizations.

The service management profile is the minimum required to use the pkgadd(1M) or pkgrm(1M) commands to add or remove software packages that contain an inventory of services in its service manifest.

Service Operator — A service operator has the ability to enable or disable any service instance on the system, as well as request that its restart or refresh method be executed. It corresponds to the `solaris.smf.manage` and `solaris.smf.modify.framework` authorizations.

Sites can define additional rights profiles customized to their needs.

Remote Repository Modification — Remote repository servers can deny modification attempts due to additional privilege checks. See NOTES.

See Also — auths(1), profiles(1), pkgadd(1M), pkgrm(1M), svccfg(1M), prof_attr(4), user_attr(4), rbac(5), smf(5)

Notes — The present version of smf(5) does not support remote repositories.

When a service is configured to be started as root but with privileges different from `limit_privileges`, the resulting process is privilege aware. This can be surprising to developers who expect `seteuid(<non-zero UID>)` to reduce privileges to basic or less.

Name smf_template – service management framework support for service metadata

Description Templates are defined by service developers to describe metadata about a service in general or individual configuration properties on a service, including human-consumable descriptions as well as definitions of valid configuration.

Administrators are provided access to templates through SMF commands that describe configuration values and validate configuration against templates.

Tool developers can use templates to provide more helpful user interfaces for service configuration.

Template Data Service metadata is defined in the template as part of the service manifest.

Consuming Template Data The `svcs -lv` and `svccfg describe` commands can be used to access metadata about properties in a human-readable format.

`svccfg(1M)`'s `validate` subcommand can be used to validate a service instance or manifest against template data. A set of `libscf(3LIB)` interfaces is available to access template data.

Template Definition The sole interface to define templates is the service manifest.

Service authors should provide template metadata including `common_names`, `descriptions`, `choices` and `constraints` for service-specific property groups and properties which they introduce. At a minimum, service authors must provide descriptions for property groups and properties in the C locale. Service authors should not provide template metadata for framework-delivered property groups such as methods and dependencies.

See the `EXAMPLES` section for an example of authoring a template definition for a service.

Template Composition All template interfaces search for template data about a property group first on the instance, then on the service, then on the service's restarter, and finally globally.

A property group template is defined by its author to apply to a specific instance, to a service and all of its instances, to a restarter's delegates, or globally. A typical service author defines the template on an instance or on a service. A template defined on an instance is applied to that instance only, and can override a template for that property group defined on the service. A template defined on the service is applied to all instances of that service.

Restarter authors can define templates in their manifest that apply to any service which uses their restarter, which is also known as a *delegate*. SMF framework authors have defined templates for property groups with well-known meanings to the entire SMF framework in the manifest for `svc:/system/svc/global`.

Templates defined globally or by the restarter and re-defined by the service or instance are flagged as a validation error. Service authors can avoid these errors by creating templates only for property groups specific to their service and not consumed by the SMF framework.

Property group templates can also be wildcarded by name or type. Only the most specific template definition applicable to a property group is honored.

Template Details

Service and Instance Templates

The `template` element defines the start of a template block. All further definitions below can be included in a template block. A `template` element can be contained in either a `service` or `instance` element. If it is contained in the `service` element, it applies to the service and all instances of that service. If it is contained in the `instance` element, it applies to only that instance of the service.

Whenever possible, we recommend defining the template data for the entire service.

```
<service ... >
   <template>
   </template>
</service>
```

Service and Instance Common Names

The entire service or instance can define a common name to describe the purpose of the service/instance.

```
<template>
  <common_name>
      <loctext xml:lang='C'>console login</loctext>
  </common_name>
<template>
```

common_name is a free-form string, but is intended to be used as a label in a GUI or CLI.

The following guidelines are recommended:

- Be brief. A word or two is usually most appropriate. Limit the name to under 40 characters.
- Be clear. The service, property group, or property name might not be helpful for humans, but *common_name* should help clarify the purpose of the entity.
- No punctuation. *common_name* is not a sentence or a paragraph. It should not contain clauses or phrases. Punctuation should only be present to meet trademark requirements.
- Capital letters must be used only for acronyms or proper names. For locales other than English, use appropriate capitalization for a sentence fragment.

Service and Instance Descriptions

The description element contains a longer description of the property group, suitable for a status line or a tool-tip:

```
<template>
  <description>
     <loctext xml:lang='C'>Provide the text login prompt on console.
     </lcotext>
  </description>
<template>
```

description Guidelines

- Use proper grammar. *description* is a sentence meant to be read by humans.
- Be brief. A few sentences are usually most appropriate.

Documentation

Documentation for this service can be defined explicitly, so that when the service is experiencing issues, or a consumer of the service wants more information on it, they can find it easily.

Documentation can include man pages or references to stable URLs for reference documentation.

```
<template>
   <documentation>
      <manpage title='ttymon' section='1M' manpath='/usr/share/man' />
      <doc_link name='docs.sun.com' uri='http://docs.sun.com' />
    </documentation>
</template>
```

Property Groups

The `pg_pattern` element contains the definitions for a property group:

```
<template>
  <pg_pattern name="pgname" type="pgtype" target="this" required="true">
  </pg_pattern>
</template>
```

name is the property group's name, and *type* is the property group's type.

target specifies what the target of this definition is. "`this`" would refer to the defining service or instance. "`instance`" can only be used in a service's template block, and means the definition applies to all instances of this service. "`delegate`" can only be used in a restarter's template block, and applies to all instances that are delegated to that restarter. "`all`", only usable by the master restarter, would refer to all services on the system. The default value of target is "`this`".

`required` indicates whether this property group is required or not. The default value of required is `false`. If `required` is `true`, both *name* and *type* must be specified.

name and/or *type* can be omitted. If either of these attributes is omitted it is treated as a wildcard. For instance, if the name attribute is omitted from the `pg_pattern` definition, the `pg_pattern` is applied to all property groups that have the specified type.

Property Group Names

The *common_name* element contains the localized, human-readable name for the property group:

```
<pg_pattern ...>
  <common_name>
    <loctext xml:lang='C'>startt method</loctext>
  </common_name>
</pg_pattern>
```

common_name is a free-form string, but is intended to be used as a label in a GUI or CLI.

The following guidelines are recommended:

- Be brief. A word or two is usually most appropriate.
- Be clear. The service, property group, or property name might not be helpful for humans, but *common_name* should help clarify the purpose of the entity.
- No punctuation. *common_name* is not a sentence or a paragraph. It should not contain clauses or phrases. Punctuation should only be present to meet trademark requirements.
- Capital letters must be used only for acronyms or proper names. For locales other than English, use appropriate capitalization for a sentence fragment.

Property Group Description

The *description* element contains a longer description of the property group, suitable for a status line or a tool-tip:

```
<pg_pattern ...>
  <description>
    <loctext xml:lang='C'>A required method which starts the service.
   </loctext>
   </description>
</pg_pattern>
```

description Guidelines

- Use proper grammar. description is a sentence meant to be read by humans.
- Be brief. A few sentences are usually most appropriate.

Properties

The *prop_pattern* element contains the definitions for a specific property:

```
<pg_pattern ...>
  <prop_pattern name="propname" type="proptype" required="true">
  </prop_pattern>
</pg_pattern>
```

name is the property's name, and *type* is the property's type.

`required` indicates whether this property is required. The default value of `required` is `false`.

name is always required. *type* is optional only if `required` is `false`.

Property Names

The *common_name* element contains the localized, human-readable name for the property:

common_name is a free-form string field, but is intended to be used as a label in a GUI or CLI.

```
<prop_pattern ...>
<common_name>
  <loctext xml:lang='C'>retry interval</loctext>
</common_name>
</prop_pattern>
```

The following guidelines are recommended:

- Be brief. A word or two is usually most appropriate.
- Be clear. The service, property group, or property name might not be helpful for humans, but *common_name* should help clarify the purpose of the entity.
- No punctuation. *common_name* is not a sentence or a paragraph. It should not contain clauses or phrases. Punctuation should only be present to meet trademark requirements.
- Capital letters must be used only for acronyms or proper names. For locales other than English, use appropriate capitalization for a sentence fragment.

Property units

The *units* element contains the localized, human-readable units for a numerical property:

```
<prop_pattern ...>
  <units>
    <loctext xml:lang='C'>seconds</loctext>
  </units>
</prop_pattern>
```

units Guidelines

The following guidelines are recommended:

- Be brief. Strive to use only a single word or label. The plural form is usually most appropriate.

- No punctuation. units is not a sentence or a paragraph. It should not contain clauses or phrases. Punctuation should only be present to meet trademark requirements.

Property description

The *description* element contains a longer description of the property, suitable for a status line or a tool-tip:

```
<prop_pattern ...>
   <description> <loctext xml:lang='C'>
    The number of seconds to wait before retry.
    </loctext> </description>
</prop_pattern>
```

description Guidelines

- Use proper grammar. *description* is a sentence meant to be read by humans.
- Be brief. A few sentences are usually most appropriate.

Property visibility

The *visibility* element specifies whether simplified views in higher level software might want to display this property.

```
<prop_pattern ...>
  <visibility value="hidden | readonly | readwrite"/>
</prop_pattern>
```

Some properties are internal implementation details and should not be presented as a configuration setting. Others might merely be read-only. This property is used to specify these restrictions. A value of hidden indicates that the property shouldn't be displayed, `readonly` means that the property isn't intended to be modified, and `readwrite` indicates the property is modifiable.

This is not a security mechanism, it is solely intended to help prevent the user from shooting himself in the foot, and to remove unnecessary clutter from CLI output or a GUI display. Hidden properties is visible in full-disclosure modes of many commands and UIs.

Property format

The *cardinality* and *internal_separators* elements constrain the structure of a property:

```
<prop_pattern ...>
  <cardinality min="1" max="1"/>
  <internal_separators>,<internal_separators>
</prop_pattern>
```

cardinality indicates the acceptable number of property values. *min* is the minimum number, and *max* is the maximum number. Both are optional. If neither is specified, `<cardinality/>` is the same as the default, zero or more values.

internal_separators specify the separator characters used for those property values into which multiple real values are packed.

Value constraints

The *constraints* element specifies what values are acceptable for a property:

```
<prop_pattern ...>
<constraints>
       <value name="blue" />
       <range min="1" max="7"/>
       <include_values type="values"/>
</constraints>
</prop_pattern>
```

The *value* element includes a possible property value. range includes an integer range.

value and *range* can be used in any combination, as restricting their use would prohibit many valid descriptions. If no value constraints are specified, the property can take on any value.

include_values includes all values specified by the values block (see `Value Descriptions` section).

Value choices

The choices block indicates which values a UI should offer the user:

```
<prop_pattern ...>
<choices>
      <range min="1" max="3"/>
      <value name="vt100" />
      <value name="xterm" />
      <include_values type="constraints"/>
      <include_values type="values"/>
</choices>
</prop_pattern>
```

range and value include ranges and individual values like they do for constraints.

include_values includes all values specified by either the constraints block or the values block (see next section).

Value Descriptions

Like property names, the values a property can take on can also have inscrutable representations. The values element contains localized, human-readable descriptions for specific property values:

```
<prop_pattern>
<values>
```

```
        <value name="blue">
                <common_name>
                        <loctext xml:lang='C'>blueloctext xml:lang='C'>blue>
                </common_name>
                <description>
                        <loctext xml:lang='C>
                                The color between green and indigo.
                        </loctext>
                </description>
        </value>
</values>
</prop_pattern>
```

common_name is a free-form string field, but is intended to be used as a label in a GUI or CLI.

The following guidelines are recommended:

- Be brief. A word or two is usually most appropriate.
- Be clear. The service, property group, or property name might not be helpful for humans, but *common_name* should help clarify the purpose of the entity.
- No punctuation. *common_name* is not a sentence or a paragraph. It should not contain clauses or phrases. Punctuation should only be present to meet trademark requirements.
- Capital letters must be used only for acronyms or proper names. For locales other than English, use appropriate capitalization for a sentence fragment.

description Guidelines

- Use proper grammar. description is a sentence meant to be read by humans.
- Be brief. A few sentences are usually most appropriate.

Examples Assuming a basic service which wants to define basic templates data looks like this:

```
<?xml version="1.0"?
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='FOOfoo:foo'>
<service name='system/foo' type='service' version='1'>
        <dependency>
                name='multi-user'
                type='service'
                grouping='require_all'
                restart_on='none'
                <service_fmri value='svc:/milestone/multi-user' />
        </dependency>
        <exec_method
                type='method'
                name='start'
                exec='/opt/foo/food'
```

```
        timeout_seconds='60'>
    </exec_method>
    <exec_method
        type='method'
        name='stop'
        exec=':kill'
        timeout_seconds='60'>
    </exec_method>
    <property_group name='config' type='application'>
        <propval name='local_only' type='boolean' value='false' />
        <propval name='config_file' type='astring'
            value='/opt/foo/foo.conf' />
    <property name='modules' type='astring'>
       <astring_list>
            <value_node value='bar'/>
            <value_node value='baz'/>
       </astring_list>
      </property>
    </property_group>

    <instance name='default' enabled='false' />
</service_bundle>
</service>
```

That service could define some basic templates data to help an administrator using this service inside of the `<service>` tags. The most helpful things are to document the purpose of the service itself and the service-specific configuration.

```
<template>
    <common_name> <loctext xml:lang='C'>
        all-purpose demonstration
    </loctext> /common_name>
    <documentation>
        <manpage title='food' section='1M'
            manpath='/opt/foo/man' />
    </documentation>

    <pg_pattern name='config' type='application' target='this'
        required='true'>
        <description> <loctext xml:lang='C'>
            Basic configuration for foo.
        </loctext> </description>
        <prop_pattern name='local_only' type='boolean'
            required='false'>
            <description> <loctext xml:lang='C'>
                Only listen to local connection requests.
            </loctext> </description>
        </prop_pattern>
```

```
                        <prop_pattern name='config_file' type='astring'
                             required='true'>
                                <cardinality min='1' max='1'/>
                                <description> <loctext xml:lang='C'>
                                     Configuration file for foo.
                                 </loctext> </description>
                         </prop_pattern>
                         <prop_pattern name='modules' type='astring'
                             required='false'>
                                 <description> <loctext xml:lang='C'>
                                     Plugin modules for foo.
                                  </loctext> /description>
                                  <values>
                                           <value name='bar'>
                                           <description> <loctext xml:lang='C'>
                                               Allow foo to access the bar.
                                            </loctext> </description>
                                            </value>
                                            <value name='baz'>
                                            <description> <loctext xml:lang='C'>
                                               Allow foo to access baz functions.
                                            </loctext> </description>
                                            </value>
                                            <value name='qux'>
                                            <description> <loctext xml:lang='C'>
                                               Allow foo to access qux functions.
                                             </loctext> </description>
                                             </value>
                                  </values>
                                  <choices>
                                      <include_values type='values'/>
                                  </choices>
                          <prop_pattern>
                </pg_pattern>
        </template>
```

Files `/usr/share/lib/xml/dtd/service_bundle.dtd.1`

See Also svcs(1), svccfg(1M), libscf(3LIB), service_bundle(4), smf(5)

Name solaris10 – Solaris 10 branded zone

Description The `solaris10` brand uses the branded zones framework described in `brands(5)` to enable Solaris 10 binary applications to run unmodified on a machine with the latest Solaris Operating System kernel.

The `solaris10` brand only supports running the Solaris 10 10/09 (Solaris 10 update 8) release, or later, within the zone.

The `solaris10` brand includes the tools necessary to install a Solaris 10 system image into a non-global zone. It also supports the tools necessary to migrate a Solaris 10 native zone to a `solaris10` branded zone. The brand supports the execution of 32-bit and 64-bit Solaris 10 applications on either SPARC or x86 machines running the latest Solaris operating system.

Configuration and Administration The `solaris10` brand supports the whole root non-global zone model. All of the required Solaris 10 software and any additional packages are installed into the private file systems of the zone.

The zone must reside on its own `zfs(1M)` dataset and only ZFS is supported. The ZFS dataset created automatically when the zone is installed or attached. If a ZFS dataset cannot be created, the zone is not installed or attached.

The `zonecfg(1M)` utility is used to configure a `solaris10` branded zone. The `SUNWsolaris10` template can be used when creating the zone or the configuration can be set up manually. Once a branded zone has been installed, that zone's brand cannot be changed or removed. The `zoneadm(1M)` utility is used to report the zone's brand type and administer the zone. The `zlogin(1)` utility is used to log in to the zone.

The support for exclusive IP-stack or delegated ZFS dataset configurations is currently experimental and has not yet been tested. Support for running these zones in a para-virtualized xVM domain is experimental and there are known problems with 64-bit x86 applications within the zone. The `/dev/sound` device cannot be configured into the branded zone. In addition, `mdb(1)` and `dtrace(1M)` are not fully functional when used in the global zone to examine processes executing within a `solaris10` branded zone.

The `solaris10` brand installer supports installing the zone from an image of an installed Solaris 10 system. This can be a full `flash_archive(4)`, `cpio(1)`, or `pax(1)` `xustar` archive. The `cpio` archive can be compressed with gzip(1) or bzip2(1). The image can also be a level 0 `ufsdump(1M)`, or a path to the top-level of a Solaris 10 system's root directory tree. The zone cannot be installed from standard Solaris 10 distribution media.

To migrate a native zone from a Solaris 10 system to the latest Solaris Operating System kernel, the attach subcommand supports installing the zone from an archive of an installed Solaris 10 native zone. As with the installer, this can be a `cpio(1)` or `pax(1)` `xustar` archive of the `zonepath`. The `cpio` archive can be compressed with gzip(1) or bzip2(1). The image can also be a path to the top-level of a Solaris 10 zone's `zonepath` directory tree. In addition to

migrating from a Solaris 10 native zone, the same migration options can be used when migrating a `solaris10` branded zone from one host to another. When migrating from Solaris 10, it is possible that the zone is configured as a `sparseroot` zone. In this case, the zone should be readied on the host before the archive is made. This ensures that the inherited directories are included in the archive.

Sub-commands The following arguments of `zoneadm(1M)` brand-specific subcommand are supported:

`attach` [`-a` *archive* | `-d` *path*]

Attach the specified Solaris 10 native zone image into the branded zone. If neither `-a` or `-d` is specified, the zone's `zonepath` is assumed to already be properly installed with the zone's files.

`-a` *archive* The path to a `cpio(1)` or `pax(1)` `xustar` archive of either an installed Solaris 10 native zone or a `solaris10` branded zone's `zonepath`. `cpio` archives can be compressed using `gzip` or `bzip2`.

`-d` *path* The path to the `zonepath` directory of either an installed Solaris 10 native zone or a `solaris10` branded zone's `zonepath`.

`install` [`-a` *archive*] [`-d` *path*] [`-p`] [`-s`] [`-u`] [`-v`]

Install the specified Solaris 10 system image into the zone. Either the `-u` or `-p` option is required *and* either the `-a` or `-d` option is required.

`-a` *archive*

The path to a `flash_archive(4)`, `cpio(1)`, `pax(1)` `xustar` archive, or a level `0` `ufsdump(1M)` of an installed Solaris 10 system. The `cpio` archives can be compressed using `gzip` or `bzip2`.

`-d` *path*

The path to the root directory of an installed Solaris 10 system.

`-p`

Preserve the system configuration after installing the zone.

`-s`

Install silently.

`-u`

Run `sys-unconfig(1M)` on the zone after installing it.

`-v`

Verbose output from the install process.

Application Support The `solaris10` zone only supports user-level Solaris 10 applications. You cannot use Solaris 10 device drivers or Solaris 10 kernel modules from inside a `solaris10` zone. However, depending on the kernel module, you might be able to use the latest Solaris kernel module version with the Solaris 10 user-level application.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWs10brandr, SUNWs10brandu
Interface Stability	Committed

See Also cpio(1), mdb(1), pax(1), zlogin(1), dtrace(1M), ufsdump(1M), sys-unconfig(1M), zfs(1M), zoneadm(1M), zonecfg(1M), flash_archive(4), attributes(5), brands(5), zones(5)

Name standards, ANSI, C, C++, ISO, POSIX, POSIX.1, POSIX.2, SUS, SUSv2, SUSv3, SVID, SVID3, XNS, XNS4, XNS5, XPG, XPG3, XPG4, XPG4v2 – standards and specifications supported by Solaris

Description Solaris 10 supports IEEE Std 1003.1 and IEEE Std 1003.2, commonly known as POSIX.1 and POSIX.2, respectively. The following table lists each version of these standards with a brief description and the SunOS or Solaris release that first conformed to it.

POSIX Standard	Description	Release
POSIX.1-1988	system interfaces and headers	SunOS 4.1
POSIX.1-1990	POSIX.1-1988 update	Solaris 2.0
POSIX.1b-1993	realtime extensions	Solaris 2.4
POSIX.1c-1996	threads extensions	Solaris 2.6
POSIX.2-1992	shell and utilities	Solaris 2.5
POSIX.2a-1992	interactive shell and utilities	Solaris 2.5
POSIX.1–2001	POSIX.1-1990, POSIX.1b-1993, POSIX.1c-1996, POSIX.2-1992, and POSIX.2a-1992 updates	Solaris 10

Solaris 10 also supports the X/Open Common Applications Environment (CAE) Portability Guide Issue 3 (XPG3) and Issue 4 (XPG4); Single UNIX Specification (SUS, also known as XPG4v2); Single UNIX Specification, Version 2 (SUSv2); and Single UNIX Specification, Version 3 (SUSv3). Both XPG4 and SUS include Networking Services Issue 4 (XNS4). SUSv2 includes Networking Services Issue 5 (XNS5).

The following table lists each X/Open specification with a brief description and the SunOS or Solaris release that first conformed to it.

X/Open CAE Specification	Description	Release
XPG3	superset of POSIX.1-1988 containing utilities from SVID3	SunOS 4.1
XPG4	superset of POSIX.1-1990, POSIX.2-1992, and POSIX.2a-1992 containing extensions to POSIX standards from XPG3	Solaris 2.4
SUS (XPG4v2)	superset of XPG4 containing historical BSD interfaces widely used by common application packages	Solaris 2.6
XNS4	sockets and XTI interfaces	Solaris 2.6

X/Open CAE Specification	Description	Release
SUSv2	superset of SUS extended to support POSIX.1b-1993, POSIX.1c-1996, and ISO/IEC 9899 (C Standard) Amendment 1	Solaris 7
XNS5	superset and LP64-clean derivative of XNS4.	Solaris 7
SUSv3	same as POSIX.1–2001	Solaris 10

The XNS4 specification is safe for use only in ILP32 (32-bit) environments and should not be used for LP64 (64-bit) application environments. Use XNS5 or SUSv3, which have LP64-clean interfaces that are portable across ILP32 and LP64 environments. Solaris releases 7 through 10 support both the ILP32 and LP64 environments.

Solaris releases 7 through 10 have been branded to conform to The Open Group's UNIX 98 Product Standard. Solaris 10 has been branded to conform to The Open Group's UNIX 03 Product Standard.

Solaris releases 2.0 through 10 support the interfaces specified by the System V Interface Definition, Third Edition, Volumes 1 through 4 (SVID3). Note, however, that since the developers of this specification (UNIX Systems Laboratories) are no longer in business and since this specification defers to POSIX and X/Open CAE specifications, there is some disagreement about what is currently required for conformance to this specification.

When `Sun Studio C Compiler 5.6` is installed, Solaris releases 2.0 through 10 support the ANSI X3.159-1989 Programming Language - C and ISO/IEC 9899:1990 Programming Language - C (C) interfaces.

When `Sun Studio C Compiler 5.6` is installed, Solaris releases 7 through 10 support ISO/IEC 9899:1990 Amendment 1:1995: C Integrity.

When `Sun Studio C Compiler 5.6` is installed, Solaris 10 supports ISO/IEC 9899:1999 Programming Languages – C.

When `Sun Studio C++ Compiler 5.6` is installed, Solaris releases 2.5.1 through 10 support ISO/IEC 14882:1998 Programming Languages - C++. Unsupported features of that standard are described in the compiler README file.

Utilities If the behavior required by POSIX.2, POSIX.2a, XPG4, SUS, or SUSv2 conflicts with historical Solaris utility behavior, the original Solaris version of the utility is unchanged; a new version that is standard-conforming has been provided in `/usr/xpg4/bin`. If the behavior required by POSIX.1–2001 or SUSv3 conflicts with historical Solaris utility behavior, a new version that is standard-conforming has been provided in `/usr/xpg4/bin` or in `/usr/xpg6/bin`. If the behavior required by POSIX.1–2001 or SUSv3 conflicts with POSIX.2, POSIX.2a, SUS, or SUSv2, a new version that is SUSv3 standard-conforming has been provided in `/usr/xpg6/bin`.

An application that wants to use standard-conforming utilitues must set the PATH (sh(1) or ksh(1)) or path (csh(1)) environment variable to specify the directories listed below in the order specified to get the appropriate utilities:

SVID3, XPG3

1. /usr/ccs/bin
2. /usr/bin
3. directory containing binaries for your compiler
4. other directories containing binaries needed by the application

POSIX.2, POSIX.2a, SUS, SUSv2, XPG4

1. /usr/xpg4/bin
2. /usr/ccs/bin
3. /usr/bin
4. directory containing binaries for your compiler
5. other directories containing binaries needed by the application

POSIX.1–2001, SUSv3

1. /usr/xpg6/bin
2. /usr/xpg4/bin
3. /usr/ccs/bin
4. /usr/bin
5. directory containing binaries for your compiler
6. other directories containing binaries needed by the application

When an application uses execlp() or execvp() (see exec(2)) to execute a shell file, or uses system(3C), the shell used to interpret the shell file depends on the standard to which the caller conforms:

Standard	Shell Used
1989 ANSI C, 1990 ISO C, 1999 ISO C, POSIX.1 (1990–2001), SUS, SUSv2, SUSv3, XPG4	/usr/xpg4/bin/sh
POSIX.1 (1988), SVID3, XPG3, no standard specified	/usr/bin/sh

Feature Test Macros

Feature test macros are used by applications to indicate additional sets of features that are desired beyond those specified by the C standard. If an application uses only those interfaces and headers defined by a particular standard (such as POSIX or X/Open CAE), then it need only define the appropriate feature test macro specified by that standard. If the application is using interfaces and headers not defined by that standard, then in addition to defining the appropriate standard feature test macro, it must also define __EXTENSIONS__. Defining __EXTENSIONS__ provides the application with access to all interfaces and headers not in conflict with the specified standard. The application must define __EXTENSIONS__ either on the compile command line or within the application source files.

1989 ANSI C, 1990 ISO C, 1999 ISO C

No feature test macros need to be defined to indicate that an application is a conforming C application.

ANSI/ISO C++

ANSI/ISO C++ does not define any feature test macros. If the standard C++ announcement macro `__cplusplus` is predefined to value 199711 or greater, the compiler operates in a standard-conforming mode, indicating C++ standards conformance. The value 199711 indicates conformance to ISO/IEC 14882:1998, as required by that standard. (As noted above, conformance to the standard is incomplete.) A standard-conforming mode is not available with compilers prior to Sun WorkShop C++ 5.0.

C++ bindings are not defined for POSIX or X/Open CAE, so specifying feature test macros such as `_POSIX_SOURCE`, `_POSIX_C_SOURCE`, and `_XOPEN_SOURCE` can result in compilation errors due to conflicting requirements of standard C++ and those specifications.

POSIX

Applications that are intended to be conforming POSIX.1 applications must define the feature test macros specified by the standard before including any headers. For the standards listed below, applications must define the feature test macros listed. Application writers must check the corresponding standards for other macros that can be queried to determine if desired options are supported by the implementation.

POSIX Standard	Feature Test Macros
POSIX.1-1990	`_POSIX_SOURCE`
POSIX.1-1990 and POSIX.2-1992 C-Language Bindings Option	`_POSIX_SOURCE` and `_POSIX_C_SOURCE=2`
POSIX.1b-1993	`_POSIX_C_SOURCE=199309L`
POSIX.1c-1996	`_POSIX_C_SOURCE=199506L`
POSIX.1–2001	`_POSIX_C_SOURCE=200112L`

SVID3

The SVID3 specification does not specify any feature test macros to indicate that an application is written to meet SVID3 requirements. The SVID3 specification was written before the C standard was completed.

X/Open CAE

To build or compile an application that conforms to one of the X/Open CAE specifications, use the following guidelines. Applications need not set the POSIX feature test macros if they require both CAE and POSIX functionality.

XPG3	The application must define `_XOPEN_SOURCE`. If `_XOPEN_SOURCE` is defined with a value, the value must be less than 500.
XPG4	The application must define `_XOPEN_SOURCE` and set `_XOPEN_VERSION=4`. If `_XOPEN_SOURCE` is defined with a value, the value must be less than 500.
SUS (XPG4v2)	The application must define `_XOPEN_SOURCE` and set `_XOPEN_SOURCE_EXTENDED=1`. If `_XOPEN_SOURCE` is defined with a value, the value must be less than 500.
SUSv2	The application must define `_XOPEN_SOURCE=500`.
SUSv3	The application must define `_XOPEN_SOURCE=600`.

Compilation A POSIX.1 (1988–1996)-, XPG4-, SUS-, or SUSv2-conforming implementation must include an ANSI X3.159-1989 (ANSI C Language) standard-conforming compilation system and the `cc` and `c89` utilities. A POSIX.1–2001– or SUSv3–conforming implementation must include an ISO/IEC 99899:1999 (1999 ISO C Language) standard-conforming compilation system and the `c99` utility. Solaris 10 was tested with the `cc`, `c89`, and `c99` utilities and the compilation environment provided by `Sun Studio C Compiler 5.6`.

When `cc` is used to link applications, `/usr/lib/values-xpg4.o` must be specified on any link/load command line, unless the application is POSIX.1–2001– or SUSv3–conforming, in which case `/usr/lib/values-xpg6.o` must be specified on any link/load compile line. The preferred way to build applications, however, is described in the table below.

An XNS4- or XNS5-conforming application must include `-l XNS` on any link/load command line in addition to defining the feature test macros specified for SUS or SUSv2, respectively.

If the compiler suppports the `redefine_extname` pragma feature (the `Sun Studio C Compiler 5.6` compilers define the macro `__PRAGMA_REDEFINE_EXTNAME` to indicate that it supports this feature), then the standard headers use `#pragma redefine_extname` directives to properly map function names onto library entry point names. This mapping provides full support for ISO C, POSIX, and X/Open namespace reservations.

If this pragma feature is not supported by the compiler, the headers use the `#define` directive to map internal function names onto appropriate library entry point names. In this instance, applications should avoid using the explicit 64-bit file offset symbols listed on the `lf64(5)` manual page, since these names are used by the implementation to name the alternative entry points.

When using `Sun Studio C Compiler 5.6` compilers, applications conforming to the specifications listed above should be compiled using the utilities and flags indicated in the following table:

```
Specification            Compiler/Flags         Feature Test Macros

_____________________________________________________________________
1989 ANSI C and 1990 ISO C    c89                none
```

1999 ISO C	c99	none
SVID3	cc -Xt -xc99=none	none
POSIX.1-1990	c89	_POSIX_SOURCE
POSIX.1-1990 and POSIX.2-1992 C-Language Bindings Option	c89	_POSIX_SOURCE and POSIX_C_SOURCE=2
POSIX.1b-1993	c89	_POSIX_C_SOURCE=199309L
POSIX.1c-1996	c89	_POSIX_C_SOURCE=199506L
POSIX.1-2001	c99	_POSIX_C_SOURCE=200112L
POSIX.1c-1996	c89	_POSIX_C_SOURCE=199506L
CAE XPG3	cc -Xa -xc99=none	_XOPEN_SOURCE
CAE XPG4	c89	_XOPEN_SOURCE and _XOPEN_VERSION=4
SUS (CAE XPG4v2) (includes XNS4)	c89	_XOPEN_SOURCE and _XOPEN_SOURCE_EXTENDED=1
SUSv2 (includes XNS5)	c89	_XOPEN_SOURCE=500
SUSv3	c99	_XOPEN_SOURCE=600

For platforms supporting the LP64 (64-bit) programming environment, SUSv2–conforming LP64 applications using XNS5 library calls should be built with command lines of the form:

```
c89 $(getconf XBS5_LP64_OFF64_CFLAGS) -D_XOPEN_SOURCE=500 \
    $(getconf XBS5_LP64_OFF64_LDFLAGS) foo.c -o foo \
    $(getconf XBS5_LP64_OFF64_LIBS) -lxnet
```

Similar SUSv3–conforming LP64 applications should be built with command lines of the form:

```
c99 $(getconf POSIX_V6_LP64_OFF64_CFLAGS) -D_XOPEN_SOURCE=600 \
    $(getconf POSIX_V6_LP64_OFF64_LDFLAGS) foo.c -o foo \
    $(getconf POSIX_V6_LP64_OFF64_LIBS) -lxnet
```

SUSv3

```
c99                         _XOPEN_SOURCE=600
```

See Also csh(1), ksh(1), sh(1), exec(2), sysconf(3C), system(3C), environ(5), lf64(5)

Name sticky – mark files for special treatment

Description The *sticky bit* (file mode bit `01000`, see chmod(2)) is used to indicate special treatment of certain files and directories. A directory for which the sticky bit is set restricts deletion of files it contains. A file in a sticky directory can only be removed or renamed by a user who has write permission on the directory, and either owns the file, owns the directory, has write permission on the file, or is a privileged user. Setting the sticky bit is useful for directories such as `/tmp`, which must be publicly writable but should deny users permission to arbitrarily delete or rename the files of others.

If the sticky bit is set on a regular file and no execute bits are set, the system's page cache will not be used to hold the file's data. This bit is normally set on swap files of diskless clients so that accesses to these files do not flush more valuable data from the system's cache. Moreover, by default such files are treated as swap files, whose inode modification times may not necessarily be correctly recorded on permanent storage.

Any user may create a sticky directory. See `chmod` for details about modifying file modes.

See Also chmod(1), chmod(2), chown(2), mkdir(2), rename(2), unlink(2)

Bugs The mkdir(2) function will not create a directory with the sticky bit set.

Name tecla, teclarc – User interface provided by the tecla library.

Description This man page describes the command-line editing features that are available to users of programs that read keyboard input via the tecla library. Users of the `tcsh` `shell` will find the default key bindings very familiar. Users of the `bash` shell will also find it quite familiar, but with a few minor differences, most notably in how forward and backward searches through the list of historical commands are performed. There are two major editing modes, one with `emacs`-like key bindings and another with `vi`-like key bindings. By default `emacs` mode is enabled, but `vi(1)` mode can alternatively be selected via the user's configuration file. This file can also be used to change the bindings of individual keys to suit the user's preferences. By default, tab completion is provided. If the application hasn't reconfigured this to complete other types of symbols, then tab completion completes file names.

Key Sequence Notation

In the rest of this man page, and also in all tecla configuration files, key sequences are expressed as follows.

`^A` or `C-a`	This is a 'CONTROL-A', entered by pressing the CONTROL key at the same time as the 'A' key.
`\\E` or `M-`	In key sequences, both of these notations can be entered either by pressing the ESCAPE key, then the following key, or by pressing the META key at the same time as the following key. Thus the key sequence `M-p` can be typed in two ways, by pressing the ESCAPE key, followed by pressing 'P', or by pressing the META key at the same time as 'P'.
up	This refers to the up-arrow key.
down	This refers to the down-arrow key.
left	This refers to the left-arrow key.
right	This refers to the right-arrow key.
a	This is just a normal 'A' key.

The Tecla Configuration File

By default, tecla looks for a file called `.teclarc` in your home directory (ie. `~/.teclarc`). If it finds this file, it reads it, interpreting each line as defining a new key binding or an editing configuration option. Since the `emacs` key-bindings are installed by default, if you want to use the non-default `vi` editing mode, the most important item to go in this file is the following line:

```
edit-mode vi
```

This will re-configure the default bindings for `vi`-mode. The complete set of arguments that this command accepts are:

vi	Install key bindings like those of the `vi` editor.
emacs	Install key bindings like those of the `emacs` editor. This is the default.

none Use just the native line editing facilities provided by the terminal driver.

To prevent the terminal bell from being rung, such as when an unrecognized control-sequence is typed, place the following line in the configuration file:

```
nobeep
```

An example of a key binding line in the configuration file is the following.

```
bind M-[2~ insert-mode
```

On many keyboards, the above key sequence is generated when one presses the insert key, so with this key binding, one can toggle between the `emacs`-mode insert and overwrite modes by hitting one key. One could also do it by typing out the above sequence of characters one by one. As explained above, the `M-` part of this sequence can be typed either by pressing the ESCAPE key before the following key, or by pressing the META key at the same time as the following key. Thus if you had set the above key binding, and the insert key on your keyboard didn't generate the above key sequence, you could still type it in either of the following 2 ways.

1. Hit the ESCAPE key momentarily, then press '[', then '2', then finally '~'.
2. Press the META key at the same time as pressing the '[' key, then press '2', then '~'.

If you set a key binding for a key sequence that is already bound to a function, the new binding overrides the old one. If in the new binding you omit the name of the new function to bind to the key sequence, the original binding becomes undefined.

Starting with versions of `libtecla` later than 1.3.3 it is now possible to bind key sequences that begin with a printable character. Previously key sequences were required to start with a CONTROL or META character.

Note that the special keywords "up", "down", "left", and "right" refer to the arrow keys, and are thus not treated as key sequences. So, for example, to rebind the up and down arrow keys to use the history search mechanism instead of the simple history recall method, you could place the following in your configuration file:

```
bind up history-search-backwards
bind down history-search-backwards
```

To unbind an existing binding, you can do this with the bind command by omitting to name any action to rebind the key sequence to. For example, by not specifying an action function, the following command unbinds the default beginning-of-line action from the `^A` key sequence:

```
bind ^A
```

If you create a `~/.teclarc` configuration file, but it appears to have no effect on the program, check the documentation of the program to see if the author chose a different name for this file.

Filename and Tilde Completion

With the default key bindings, pressing the TAB key (aka. `^I`) results in tecla attempting to complete the incomplete file name that precedes the cursor. Tecla searches backwards from the cursor, looking for the start of the file name, stopping when it hits either a space or the start of the line. If more than one file has the specified prefix, then tecla completes the file name up to the point at which the ambiguous matches start to differ, then lists the possible matches.

In addition to literally written file names, tecla can complete files that start with `~/` and `~user/` expressions and that contain `$envvar` expressions. In particular, if you hit TAB within an incomplete `~user`, expression, tecla will attempt to complete the username, listing any ambiguous matches.

The completion binding is implemented using the `cpl_complete_word()` function, which is also available separately to users of this library. See the `cpl_complete_word(3TECLA)` man page for more details.

Filename Expansion

With the default key bindings, pressing `^X*` causes tecla to expand the file name that precedes the cursor, replacing `~/` and `~user/` expressions with the corresponding home directories, and replacing `$envvar` expressions with the value of the specified environment variable, then if there are any wildcards, replacing the so far expanded file name with a space-separated list of the files which match the wild cards.

The expansion binding is implemented using the `ef_expand_file()` function. See the `ef_expand_file(3TECLA)` man page for more details.

Recalling Previously Typed Lines

Every time that a new line is entered by the user, it is appended to a list of historical input lines maintained within the `GetLine` resource object. You can traverse up and down this list using the up and down arrow keys. Alternatively, you can do the same with the `^P`, and `^N` keys, and in `vi` command mode you can alternatively use the k and j characters. Thus pressing up-arrow once, replaces the current input line with the previously entered line. Pressing up-arrow again, replaces this with the line that was entered before it, etc.. Having gone back one or more lines into the history list, one can return to newer lines by pressing down-arrow one or more times. If you do this sufficient times, you will return to the original line that you were entering when you first hit up-arrow.

Note that in `vi` mode, all of the history recall functions switch the library into command mode.

In `emacs` mode the `M-p` and `M-n` keys work just like the `^P` and `^N` keys, except that they skip all but those historical lines which share the prefix that precedes the cursor. In `vi` command mode the upper case 'K' and 'J' characters do the same thing, except that the string that they search for includes the character under the cursor as well as what precedes it.

Thus for example, suppose that you were in `emacs` mode, and you had just entered the following list of commands in the order shown:

```
ls ~/tecla/
cd ~/tecla
ls -l getline.c
emacs ~/tecla/getline.c
```

If you next typed:

```
ls
```

and then hit `M-p`, then rather than returning the previously typed `emacs` line, which doesn't start with "ls", tecla would recall the "ls -l getline.c" line. Pressing `M-p` again would recall the "ls ~/tecla/" line.

Note that if the string that you are searching for, contains any of the special characters, `*`, `?`, or '[', then it is interpretted as a pattern to be matched. Thus, cotinuing with the above example, after typing in the list of commands shown, if you then typed:

```
*tecla*
```

and hit `M-p`, then the "`emacs` ~/tecla/getline.c" line would be recalled first, since it contains the word tecla somewhere in the line, Similarly, hitting `M-p` again, would recall the "ls ~/tecla/" line, and hitting it once more would recall the "ls ~/tecla/" line. The pattern syntax is the same as that described for file name expansion, in the `ef_expand_file`(3TECLA).

History Files

Authors of programs that use the tecla library have the option of saving historical command-lines in a file before exiting, and subsequently reading them back in from this file when the program is next started. There is no standard name for this file, since it makes sense for each application to use its own history file, so that commands from different applications don't get mixed up.

International Character Sets

Since `libtecla` version 1.4.0, tecla has been 8-bit clean. This means that all 8-bit characters that are printable in the user's current locale are now displayed verbatim and included in the returned input line. Assuming that the calling program correctly contains a call like the following,

```
setlocale(LC_CTYPE, "");
```

then the current locale is determined by the first of the environment variables `LC_CTYPE`, `LC_ALL`, and `LANG`, that is found to contain a valid locale name. If none of these variables are defined, or the program neglects to call `setlocale`, then the default C locale is used, which is US 7-bit ASCII. On most unix-like platforms, you can get a list of valid locales by typing the command:

```
locale -a
```

at the shell prompt.

Meta Keys and Locales

Beware that in most locales other than the default C locale, META characters become printable, and they are then no longer considered to match `M-c` style key bindings. This allows international characters to be entered with the compose key without unexpectedly triggering META key bindings. You can still invoke META bindings, since there are actually two ways to do this. For example the binding `M-c` can also be invoked by pressing the ESCAPE key momentarily, then pressing the c key, and this will work regardless of locale. Moreover, many modern terminal emulators, such as gnome's gnome-terminal's and KDE's konsole terminals, already generate escape pairs like this when you use the META key, rather than a real meta character, and other emulators usually have a way to request this behavior, so you can continue to use the META key on most systems.

For example, although xterm terminal emulators generate real 8-bit meta characters by default when you use the META key, they can be configured to output the equivalent escape pair by setting their `EightBitInput` X resource to False. You can either do this by placing a line like the following in your `~/.Xdefaults` file,

```
XTerm*EightBitInput: False
```

or by starting an `xterm` with an `-xrm` '*EightBitInput: False' command-line argument. In recent versions of xterm you can toggle this feature on and off with the 'Meta Sends Escape' option in the menu that is displayed when you press the left mouse button and the CONTROL key within an xterm window. In CDE, dtterms can be similarly coerced to generate escape pairs in place of meta characters, by setting the `Dtterm*KshMode` resource to True.

Entering International Characters

If you don't have a keyboard that generates all of the international characters that you need, there is usually a compose key that will allow you to enter special characters, or a way to create one. For example, under X windows on unix-like systems, if your keyboard doesn't have a compose key, you can designate a redundant key to serve this purpose with the xmodmap command. For example, on many PC keyboards there is a microsoft-windows key, which is otherwise useless under Linux. On a laptop, for example, the `xev` program might report that pressing this key generates keycode 115. To turn this key into a COMPOSE key, do the following:

```
xmodmap -e 'keycode 115 = Multi_key'
```

Type this key followed by a " character to enter an 'I' with a umlaut over it.

The Available Key Binding Functions

The following is a list of the editing functions provided by the tecla library. The names in the leftmost column of the list can be used in configuration files to specify which function a given key or combination of keys should invoke. They are also used in the next two sections to list the default key bindings in `emacs` and `vi` modes.

user-interrupt	Send a SIGINT signal to the parent process.
suspend	Suspend the parent process.

stop-output	Pause terminal output.
start-output	Resume paused terminal output.
literal-next	Arrange for the next character to be treated as a normal character. This allows control characters to be entered.
cursor-right	Move the cursor one character right.
cursor-left	Move the cursor one character left.
insert-mode	Toggle between insert mode and overwrite mode.
beginning-of-line	Move the cursor to the beginning of the line.
end-of-line	Move the cursor to the end of the line.
delete-line	Delete the contents of the current line.
kill-line	Delete everything that follows the cursor.
backward-kill-line	Delete all characters between the cursor and the start of the line.
forward-word	Move to the end of the word which follows the cursor.
forward-to-word	Move the cursor to the start of the word that follows the cursor.
backward-word	Move to the start of the word which precedes the cursor.
goto-column	Move the cursor to the 1-relative column in the line specified by any preceding digit-argument sequences (see Entering Repeat Counts below).
find-parenthesis	If the cursor is currently over a parenthesis character, move it to the matching parenthesis character. If not over a parenthesis character move right to the next close parenthesis.
forward-delete-char	Delete the character under the cursor.
backward-delete-char	Delete the character which precedes the cursor.
list-or-eof	This is intended for binding to `^D`. When invoked when the cursor is within the line it displays all possible completions then redisplays the line unchanged. When invoked on an empty line, it signals end-of-input (EOF) to the caller of `gl_get_line()`.
del-char-or-list-or-eof	This is intended for binding to `^D`. When invoked when the cursor is within the line it invokes forward-delete-char. When invoked at the end of the line it displays all possible

	completions then redisplays the line unchanged. When invoked on an empty line, it signals end-of-input (EOF) to the caller of `gl_get_line()`.
forward-delete-word	Delete the word which follows the cursor.
backward-delete-word	Delete the word which precedes the cursor.
upcase-word	Convert all of the characters of the word which follows the cursor, to upper case.
downcase-word	Convert all of the characters of the word which follows the cursor, to lower case.
capitalize-word	Capitalize the word which follows the cursor.
change-case	If the next character is upper case, toggle it to lower case and vice versa.
redisplay	Redisplay the line.
clear-screen	Clear the terminal, then redisplay the current line.
transpose-chars	Swap the character under the cursor with the character just before the cursor.
set-mark	Set a mark at the position of the cursor.
exchange-point-and-mark	Move the cursor to the last mark that was set, and move the mark to where the cursor used to be.
kill-region	Delete the characters that lie between the last mark that was set, and the cursor.
copy-region-as-kill	Copy the text between the mark and the cursor to the cut buffer, without deleting the original text.
yank	Insert the text that was last deleted, just before the current position of the cursor.
append-yank	Paste the current contents of the cut buffer, after the cursor.
up-history	Recall the next oldest line that was entered. Note that in `vi` mode you are left in command mode.
down-history	Recall the next most recent line that was entered. If no history recall session is currently active, the next line from a previous recall session is recalled. Note that in vi mode you are left in command mode.
history-search-backward	Recall the next oldest line who's prefix matches the string which currently precedes the cursor (in `vi` command-mode

	the character under the cursor is also included in the search string). Note that in `vi` mode you are left in command mode.
history-search-forward	Recall the next newest line who's prefix matches the string which currently precedes the cursor (in `vi` command-mode the character under the cursor is also included in the search string). Note that in `vi` mode you are left in command mode.
history-re-search-backward	Recall the next oldest line who's prefix matches that established by the last invocation of either history-search-forward or history-search-backward.
history-re-search-forward	Recall the next newest line who's prefix matches that established by the last invocation of either history-search-forward or history-search-backward.
complete-word	Attempt to complete the incomplete word which precedes the cursor. Unless the host program has customized word completion, file name completion is attempted. In `vi` commmand mode the character under the cursor is also included in the word being completed, and you are left in `vi` insert mode.
expand-filename	Within the command line, expand wild cards, tilde expressions and dollar expressions in the file name which immediately precedes the cursor. In `vi` commmand mode the character under the cursor is also included in the file name being expanded, and you are left in `vi` insert mode.
list-glob	List any file names which match the wild-card, tilde and dollar expressions in the file name which immediately precedes the cursor, then redraw the input line unchanged.
list-history	Display the contents of the history list for the current history group. If a repeat count of `> 1` is specified, only that many of the most recent lines are displayed. See the Entering Repeat Counts section.
read-from-file	Temporarily switch to reading input from the file who's name precedes the cursor.
read-init-files	Re-read `teclarc` configuration files.
beginning-of-history	Move to the oldest line in the history list. Note that in `vi` mode you are left in command mode.
end-of-history	Move to the newest line in the history list (ie. the current line). Note that in `vi` mode this leaves you in command mode.

digit-argument	Enter a repeat count for the next key binding function. For details, see the Entering Repeat Counts section.
newline	Terminate and return the current contents of the line, after appending a newline character. The newline character is normally '\ ', but will be the first character of the key sequence that invoked the newline action, if this happens to be a printable character. If the action was invoked by the '\ ' newline character or the '\\r' carriage return character, the line is appended to the history buffer.
repeat-history	Return the line that is being edited, then arrange for the next most recent entry in the history buffer to be recalled when tecla is next called. Repeatedly invoking this action causes successive historical input lines to be re-executed. Note that this action is equivalent to the 'Operate' action in ksh.
ring-bell	Ring the terminal bell, unless the bell has been silenced via the nobeep configuration option (see The Tecla Configuration File section).
forward-copy-char	Copy the next character into the cut buffer (NB. use repeat counts to copy more than one).
backward-copy-char	Copy the previous character into the cut buffer.
forward-copy-word	Copy the next word into the cut buffer.
backward-copy-word	Copy the previous word into the cut buffer.
forward-find-char	Move the cursor to the next occurrence of the next character that you type.
backward-find-char	Move the cursor to the last occurrence of the next character that you type.
forward-to-char	Move the cursor to the character just before the next occurrence of the next character that the user types.
backward-to-char	Move the cursor to the character just after the last occurrence before the cursor of the next character that the user types.
repeat-find-char	Repeat the last backward-find-char, forward-find-char, backward-to-char or forward-to-char.
invert-refind-char	Repeat the last backward-find-char, forward-find-char, backward-to-char, or forward-to-char in the opposite direction.

delete-to-column	Delete the characters from the cursor up to the column that is specified by the repeat count.
delete-to-parenthesis	Delete the characters from the cursor up to and including the matching parenthesis, or next close parenthesis.
forward-delete-find	Delete the characters from the cursor up to and including the following occurence of the next character typed.
backward-delete-find	Delete the characters from the cursor up to and including the preceding occurence of the next character typed.
forward-delete-to	Delete the characters from the cursor up to, but not including, the following occurence of the next character typed.
backward-delete-to	Delete the characters from the cursor up to, but not including, the preceding occurence of the next character typed.
delete-refind	Repeat the last *-delete-find or *-delete-to action.
delete-invert-refind	Repeat the last *-delete-find or *-delete-to action, in the opposite direction.
copy-to-column	Copy the characters from the cursor up to the column that is specified by the repeat count, into the cut buffer.
copy-to-parenthesis	Copy the characters from the cursor up to and including the matching parenthesis, or next close parenthesis, into the cut buffer.
forward-copy-find	Copy the characters from the cursor up to and including the following occurence of the next character typed, into the cut buffer.
backward-copy-find	Copy the characters from the cursor up to and including the preceding occurence of the next character typed, into the cut buffer.
forward-copy-to	Copy the characters from the cursor up to, but not including, the following occurence of the next character typed, into the cut buffer.
backward-copy-to	Copy the characters from the cursor up to, but not including, the preceding occurence of the next character typed, into the cut buffer.
copy-refind	Repeat the last *-copy-find or *-copy-to action.

copy-invert-refind	Repeat the last *-copy-find or *-copy-to action, in the opposite direction.
vi-mode	Switch to `vi` mode from emacs mode.
emacs-mode	Switch to `emacs` mode from `vi` mode.
vi-insert	From `vi` command mode, switch to insert mode.
vi-overwrite	From `vi` command mode, switch to overwrite mode.
vi-insert-at-bol	From `vi` command mode, move the cursor to the start of the line and switch to insert mode.
vi-append-at-eol	From `vi` command mode, move the cursor to the end of the line and switch to append mode.
vi-append	From `vi` command mode, move the cursor one position right, and switch to insert mode.
vi-replace-char	From `vi` command mode, replace the character under the cursor with the next character entered.
vi-forward-change-char	From `vi` command mode, delete the next character then enter insert mode.
vi-backward-change-char	From vi command mode, delete the preceding character then enter insert mode.
vi-forward-change-word	From `vi` command mode, delete the next word then enter insert mode.
vi-backward-change-word	From vi command mode, delete the preceding word then enter insert mode.
vi-change-rest-of-line	From `vi` command mode, delete from the cursor to the end of the line, then enter insert mode.
vi-change-line	From `vi` command mode, delete the current line, then enter insert mode.
vi-change-to-bol	From `vi` command mode, delete all characters between the cursor and the beginning of the line, then enter insert mode.
vi-change-to-column	From `vi` command mode, delete the characters from the cursor up to the column that is specified by the repeat count, then enter insert mode.
vi-change-to-parenthesis	Delete the characters from the cursor up to and including the matching parenthesis, or next close parenthesis, then enter `vi` insert mode.

vi-forward-change-find	From `vi` command mode, delete the characters from the cursor up to and including the following occurence of the next character typed, then enter insert mode.
vi-backward-change-find	From vi command mode, delete the characters from the cursor up to and including the preceding occurence of the next character typed, then enter insert mode.
vi-forward-change-to	From `vi` command mode, delete the characters from the cursor up to, but not including, the following occurence of the next character typed, then enter insert mode.
vi-backward-change-to	From `vi` command mode, delete the characters from the cursor up to, but not including, the preceding occurence of the next character typed, then enter insert mode.
vi-change-refind	Repeat the last vi-*-change-find or vi-*-change-to action.
vi-change-invert-refind	Repeat the last vi-*-change-find or vi-*-change-to action, in the opposite direction.
vi-undo	In `vi` mode, undo the last editing operation.
vi-repeat-change	In `vi` command mode, repeat the last command that modified the line.

Default Key Bindings In `emacs` Mode

The following default key bindings, which can be overriden by the tecla configuration file, are designed to mimic most of the bindings of the unix `tcsh` `shell` shell, when it is in `emacs` editing mode.

This is the default editing mode of the tecla library.

Under UNIX the terminal driver sets a number of special keys for certain functions. The tecla library attempts to use the same key bindings to maintain consistency. The key sequences shown for the following 6 bindings are thus just examples of what they will probably be set to. If you have used the stty command to change these keys, then the default bindings should match.

`^C`	user-interrupt
`^\\`	abort
`^Z`	suspend
`^Q`	start-output
`^S`	stop-output
`^V`	literal-next

The cursor keys are refered to by name, as follows. This is necessary because different types of terminals generate different key sequences when their cursor keys are pressed.

right	cursor-right
left	cursor-left
up	up-history
down	down-history

The remaining bindings don't depend on the terminal setttings.

`^F`	cursor-right
`^B`	cursor-left
`M-i`	insert-mode
`^A`	beginning-of-line
`^E`	end-of-line
`^U`	delete-line
`^K`	kill-line
`M-f`	forward-word
`M-b`	backward-word
`^D`	del-char-or-list-or-eof
`^H`	backward-delete-char
`^?`	backward-delete-char
`M-d`	forward-delete-word
`M-^H`	backward-delete-word
`M-^?`	backward-delete-word
`M-u`	upcase-word
`M-l`	downcase-word
`M-c`	capitalize-word
`^R`	redisplay
`^L`	clear-screen
`^T`	transpose-chars
`^@`	set-mark

^X^X	exchange-point-and-mark
^W	kill-region
M-w	copy-region-as-kill
^Y	yank
^P	up-history
^N	down-history
M-p	history-search-backward
M-n	history-search-forward
^I	complete-word
^X*	expand-filename
^X^F	read-from-file
^X^R	read-init-files
^Xg	list-glob
^Xh	list-history
M-<	beginning-of-history
M->	end-of-history
\	newline
\\r	newline
M-o	repeat-history
M-^V	vi-mode
M-0, M-1, ... M-9	digit-argument (see below)

Note that ^I is what the TAB key generates, and that ^@ can be generated not only by pressing the CONTROL key and the @ key simultaneously, but also by pressing the CONTROL key and the space bar at the same time.

Default Key Bindings in vi Mode

The following default key bindings are designed to mimic the vi style of editing as closely as possible. This means that very few editing functions are provided in the initial character input mode, editing functions instead being provided by the vi command mode. The vi command mode is entered whenever the ESCAPE character is pressed, or whenever a key sequence that starts with a meta character is entered. In addition to mimicing vi, libtecla provides bindings for tab completion, wild-card expansion of file names, and historical line recall.

To learn how to tell the tecla library to use `vi` mode instead of the default `emacs` editing mode, see the earlier section entitled The Tecla Configuration File.

Under UNIX the terminal driver sets a number of special keys for certain functions. The tecla library attempts to use the same key bindings to maintain consistency, binding them both in input mode and in command mode. The key sequences shown for the following 6 bindings are thus just examples of what they will probably be set to. If you have used the `stty` command to change these keys, then the default bindings should match.

`^C`	user-interrupt
`^\\`	abort
`^Z`	suspend
`^Q`	start-output
`^S`	stop-output
`^V`	literal-next
`M-^C`	user-interrupt
`M-^\\`	abort
`M-^Z`	suspend
`M-^Q`	start-output
`M-^S`	stop-output

Note that above, most of the bindings are defined twice, once as a raw control code like `^C` and then a second time as a META character like `M-^C`. The former is the binding for `vi` input mode, whereas the latter is the binding for `vi` command mode. Once in command mode all key sequences that the user types that they don't explicitly start with an ESCAPE or a META key, have their first key secretly converted to a META character before the key sequence is looked up in the key binding table. Thus, once in command mode, when you type the letter `i`, for example, the tecla library actually looks up the binding for `M-i`.

The cursor keys are refered to by name, as follows. This is necessary because different types of terminals generate different key sequences when their cursor keys are pressed.

`right`	cursor-right
`left`	cursor-left
`up`	up-history
`down`	down-history

The cursor keys normally generate a key sequence that start with an ESCAPE character, so beware that using the arrow keys will put you into command mode (if you aren't already in command mode).

The following are the terminal-independent key bindings for `vi` input mode.

`^D`	list-or-eof
`^G`	list-glob
`^H`	backward-delete-char
`^I`	complete-word
`\\r`	newline
`\`	newline
`^L`	clear-screen
`^N`	down-history
`^P`	up-history
`^R`	redisplay
`^U`	backward-kill-line
`^W`	backward-delete-word
`^X*`	expand-filename
`^X^F`	read-from-file
`^X^R`	read-init-files
`^?`	backward-delete-char

The following are the key bindings that are defined in `vi` command mode, this being specified by them all starting with a META character. As mentioned above, once in command mode the initial meta character is optional. For example, you might enter command mode by typing ESCAPE, and then press 'H' twice to move the cursor two positions to the left. Both 'H' characters get quietly converted to `M-h` before being compared to the key binding table, the first one because ESCAPE followed by a character is always converted to the equivalent META character, and the second because command mode was already active.

`M-\\`	cursor-right (META-space)
`M-$`	end-of-line
`M-*`	expand-filename
`M-+`	down-history

M--	up-history
M-<	beginning-of-history
M->	end-of-history
M-^	beginning-of-line
M-	repeat-find-char
M-,	invert-refind-char
M-|	goto-column
M-~	change-case
M-.	vi-repeat-change
M-%	find-parenthesis
M-a	vi-append
M-A	vi-append-at-eol
M-b	backward-word
M-B	backward-word
M-C	vi-change-rest-of-line
M-cb	vi-backward-change-word
M-cB	vi-backward-change-word
M-cc	vi-change-line
M-ce	vi-forward-change-word
M-cE	vi-forward-change-word
M-cw	vi-forward-change-word
M-cW	vi-forward-change-word
M-cF	vi-backward-change-find
M-cf	vi-forward-change-find
M-cT	vi-backward-change-to
M-ct	vi-forward-change-to
M-c;	vi-change-refind
M-c,	vi-change-invert-refind
M-ch	vi-backward-change-char

M-c^H	vi-backward-change-char
M-c^?	vi-backward-change-char
M-cl	vi-forward-change-char
M-c\\	vi-forward-change-char (META-c-space)
M-c^	vi-change-to-bol
M-c0	vi-change-to-bol
M-c$	vi-change-rest-of-line
M-c|	vi-change-to-column
M-c%	vi-change-to-parenthesis
M-dh	backward-delete-char
M-d^H	backward-delete-char
M-d^?	backward-delete-char
M-dl	forward-delete-char
M-d	forward-delete-char (META-d-space)
M-dd	delete-line
M-db	backward-delete-word
M-dB	backward-delete-word
M-de	forward-delete-word
M-dE	forward-delete-word
M-dw	forward-delete-word
M-dW	forward-delete-word
M-dF	backward-delete-find
M-df	forward-delete-find
M-dT	backward-delete-to
M-dt	forward-delete-to
M-d;	delete-refind
M-d,	delete-invert-refind
M-d^	backward-kill-line
M-d0	backward-kill-line

`M-d$`	kill-line
`M-D`	kill-line
`M-d|`	delete-to-column
`M-d%`	delete-to-parenthesis
`M-e`	forward-word
`M-E`	forward-word
`M-f`	forward-find-char
`M-F`	backward-find-char
`M--`	up-history
`M-h`	cursor-left
`M-H`	beginning-of-history
`M-i`	vi-insert
`M-I`	vi-insert-at-bol
`M-j`	down-history
`M-J`	history-search-forward
`M-k`	up-history
`M-K`	history-search-backward
`M-l`	cursor-right
`M-L`	end-of-history
`M-n`	history-re-search-forward
`M-N`	history-re-search-backward
`M-p`	append-yank
`M-P`	yank
`M-r`	vi-replace-char
`M-R`	vi-overwrite
`M-s`	vi-forward-change-char
`M-S`	vi-change-line
`M-t`	forward-to-char
`M-T`	backward-to-char

M-u	vi-undo
M-w	forward-to-word
M-W	forward-to-word
M-x	forward-delete-char
M-X	backward-delete-char
M-yh	backward-copy-char
M-y^H	backward-copy-char
M-y^?	backward-copy-char
M-yl	forward-copy-char
M-y\\	forward-copy-char (META-y-space)
M-ye	forward-copy-word
M-yE	forward-copy-word
M-yw	forward-copy-word
M-yW	forward-copy-word
M-yb	backward-copy-word
M-yB	backward-copy-word
M-yf	forward-copy-find
M-yF	backward-copy-find
M-yt	forward-copy-to
M-yT	backward-copy-to
M-y;	copy-refind
M-y,	copy-invert-refind
M-y^	copy-to-bol
M-y0	copy-to-bol
M-y$	copy-rest-of-line
M-yy	copy-line
M-Y	copy-line
M-y|	copy-to-column
M-y%	copy-to-parenthesis

`M-^E`	emacs-mode
`M-^H`	cursor-left
`M-^?`	cursor-left
`M-^L`	clear-screen
`M-^N`	down-history
`M-^P`	up-history
`M-^R`	redisplay
`M-^D`	list-or-eof
`M-^I`	complete-word
`M-\\r`	newline
`M-\`	newline
`M-^X^R`	read-init-files
`M-^Xh`	list-history
`M-0, M-1, ... M-9`	digit-argument (see below)

Note that `^I` is what the TAB key generates.

Entering Repeat Counts

Many of the key binding functions described previously, take an optional count, typed in before the target key sequence. This is interpreted as a repeat count by most bindings. A notable exception is the goto-column binding, which interprets the count as a column number.

By default you can specify this count argument by pressing the META key while typing in the numeric count. This relies on the digit-argument action being bound to 'META-0', 'META-1' etc. Once any one of these bindings has been activated, you can optionally take your finger off the META key to type in the rest of the number, since every numeric digit thereafter is treated as part of the number, unless it is preceded by the literal-next binding. As soon as a non-digit, or literal digit key is pressed the repeat count is terminated and either causes the just typed character to be added to the line that many times, or causes the next key binding function to be given that argument.

For example, in `emacs` mode, typing:

```
M-12a
```

causes the letter 'a' to be added to the line 12 times, whereas

```
M-4M-c
```

Capitalizes the next 4 words.

In `vi` command mode the meta modifier is automatically added to all characters typed in, so to enter a count in `vi` command-mode, just involves typing in the number, just as it does in the `vi` editor itself. So for example, in vi command mode, typing:

```
4w2x
```

moves the cursor four words to the right, then deletes two characters.

You can also bind digit-argument to other key sequences. If these end in a numeric digit, that digit gets appended to the current repeat count. If it doesn't end in a numeric digit, a new repeat count is started with a value of zero, and can be completed by typing in the number, after letting go of the key which triggered the digit-argument action.

Files

`/usr/lib/libtecla.so`	The tecla library
`/usr/include/libtecla.h`	The tecla header file
`~/.teclarc`	The personal tecla customization file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtecla
Interface Stability	Evolving

See Also `vi(1)`, `cpl_complete_word(3TECLA)`, `ef_expand_file(3TECLA)`, `gl_get_line(3TECLA)`, `gl_io_mode(3TECLA)`, `libtecla(3LIB)`, `pca_lookup_file(3TECLA)`, `attributes(5)`

Name term – conventional names for terminals

Description Terminal names are maintained as part of the shell environment in the environment variable `TERM`. See sh(1), profile(4), and environ(5). These names are used by certain commands (for example, `tabs`, `tput`, and `vi`) and certain functions (for example, see curses(3CURSES)).

Files under `/usr/share/lib/terminfo` are used to name terminals and describe their capabilities. These files are in the format described in terminfo(4). Entries in `terminfo` source files consist of a number of comma-separated fields. To print a description of a terminal *term*, use the command `infocmp -I` *term*. See infocmp(1M). White space after each comma is ignored. The first line of each terminal description in the `terminfo` database gives the names by which `terminfo` knows the terminal, separated by bar (`|`) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable `TERMINFO` in `$HOME/.profile`; see profile(4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, `att4425`. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from the set `a` through `z` and `0` through `9`, make up a basic terminal name. Names should generally be based on original vendors rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode is `att4425`–`w`. The following suffixes should be used where possible:

Suffix	Meaning	Example
–w	Wide mode (more than 80 columns)	att4425–w
–am	With auto. margins (usually default)	vt100–am
–nam	Without automatic margins	vt100–nam
–*n*	Number of lines on the screen	aaa–60
–na	No arrow keys (leave them in local)	c100–na
–np	Number of pages of memory	c100–4p
–rv	Reverse video	att4415–rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (for example, `-w`), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the `terminfo(4)` database unique. Terminal entries that are present only for inclusion in other entries via the `use=` facilities should have a '`+`' in their name, as in `4415+nl`.

Here are some of the known terminal names: (For a complete list, enter the command `ls -C /usr/share/lib/terminfo/?`).

2621,hp2621	Hewlett-Packard 2621 series
2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer, compressed mode
2631–e	Hewlett-Packard 2631 line printer, expanded mode
2640,hp2640	Hewlett-Packard 2640 series
2645,hp2645	Hewlett-Packard 2645 series
3270	IBM Model 3270
33,tty33	AT&T Teletype Model 33 KSR
35,tty35	AT&T Teletype Model 35 KSR
37,tty37	AT&T Teletype Model 37 KSR
4000a	Trendata 4000a
4014,tek4014	TEKTRONIX 4014
40,tty40	AT&T Teletype Dataspeed 40/2
43,tty43	AT&T Teletype Model 43 KSR
4410,5410	AT&T 4410/5410 in 80-column mode, version 2
4410–nfk,5410–nfk	AT&T 4410/5410 without function keys, version 1
4410–nsl,5410–nsl	AT&T 4410/5410 without pln defined
4410–w,5410–w	AT&T 4410/5410 in 132-column mode
4410v1,5410v1	AT&T 4410/5410 in 80-column mode, version 1
4410v1–w,5410v1–w	AT&T 4410/5410 in 132-column mode, version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415–nl,5420–nl	AT&T 4415/5420 without changing labels
4415–rv,5420–rv	AT&T 4415/5420 80 columns in reverse video

4415–rv–nl,5420–rv–nl	AT&T 4415/5420 reverse video without changing labels
4415–w,5420–w	AT&T 4415/5420 in 132-column mode
4415–w–nl,5420–w–nl	AT&T 4415/5420 in 132-column mode without changing labels
4415–w–rv,5420–w–rv	AT&T 4415/5420 132 columns in reverse video
4418,5418	AT&T 5418 in 80-column mode
4418–w,5418–w	AT&T 5418 in 132-column mode
4420	AT&T Teletype Model 4420
4424	AT&T Teletype Model 4424
4424-2	AT&T Teletype Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425–fk,5425–fk	AT&T 4425/5425 without function keys
4425–nl,5425–nl	AT&T 4425/5425 without changing labels in 80-column mode
4425–w,5425–w	AT&T 4425/5425 in 132-column mode
4425–w–fk,5425–w–fk	AT&T 4425/5425 without function keys in 132-column mode
4425–nl–w,5425–nl–w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T Teletype Model 4426S
450	DASI 450 (same as Diablo 1620)
450–12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2–w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620–24,dmd–24	AT&T Teletype Model DMD 5620 in a 24x80 layer
5620–34,dmd–34	AT&T Teletype Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610–w,610bct–w	AT&T 610 bct terminal in 132-column mode

630,630MTG	AT&T 630 Multi-Tasking Graphics terminal
7300,pc7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505–24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous Teletype Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form `-T`*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable `TERM`, which, in turn, should contain *term*.

Files `/usr/share/lib/terminfo/?/*` compiled terminal description database

See Also `sh(1)`, `stty(1)`, `tabs(1)`, `tput(1)`, `vi(1)`, `infocmp(1M)`, `curses(3CURSES)`, `profile(4)`, `terminfo(4)`, `environ(5)`

Name threads, pthreads – POSIX pthreads and Solaris threads concepts

Synopsis

POSIX
```
cc –mt [ flag... ] file... [ -lrt library... ]
#include <pthread.h>
```

Solaris
```
cc –mt [ flag... ] file... [ library... ]
#include <sched.h>
#include <thread.h>
```

Description POSIX and Solaris threads each have their own implementation within libc(3LIB). Both implementations are interoperable, their functionality similar, and can be used within the same application. Only POSIX threads are guaranteed to be fully portable to other POSIX-compliant environments. POSIX and Solaris threads require different source, include files and linking libraries. See SYNOPSIS.

Similarities Most of the POSIX and Solaris threading functions have counterparts with each other. POSIX function names, with the exception of the semaphore names, have a "pthread" prefix. Function names for similar POSIX and Solaris functions have similar endings. Typically, similar POSIX and Solaris functions have the same number and use of arguments.

Differences POSIX pthreads and Solaris threads differ in the following ways:

- POSIX threads are more portable.
- POSIX threads establish characteristics for each thread according to configurable attribute objects.
- POSIX pthreads implement thread cancellation.
- POSIX pthreads enforce scheduling algorithms.
- POSIX pthreads allow for clean-up handlers for fork(2) calls.
- Solaris threads can be suspended and continued.
- Solaris threads implement daemon threads, for whose demise the process does not wait.

Function Comparison The following table compares the POSIX pthreads and Solaris threads functions. When a comparable interface is not available either in POSIX pthreads or Solaris threads, a hyphen (–) appears in the column.

Functions Related to Creation

POSIX	Solaris
pthread_create()	thr_create()
pthread_attr_init()	–

pthread_attr_setdetachstate()	–
pthread_attr_getdetachstate()	–
pthread_attr_setinheritsched()	–
pthread_attr_getinheritsched()	–
pthread_attr_setschedparam()	–
pthread_attr_getschedparam()	–
pthread_attr_setschedpolicy()	–
pthread_attr_getschedpolicy()	–
pthread_attr_setscope()	–
pthread_attr_getscope()	–
pthread_attr_setstackaddr()	–
pthread_attr_getstackaddr()	–
pthread_attr_setstacksize()	–
pthread_attr_getstacksize()	–
pthread_attr_getguardsize()	–
pthread_attr_setguardsize()	–
pthread_attr_destroy()	–
–	thr_min_stack()

Functions Related to Exit

POSIX	Solaris
pthread_exit()	thr_exit()
pthread_join()	thr_join()
pthread_detach()	–

Functions Related to Thread Specific Data

POSIX	Solaris
pthread_key_create()	thr_keycreate()
pthread_setspecific()	thr_setspecific()
pthread_getspecific()	thr_getspecific()
pthread_key_delete()	–

Functions Related to Signals

POSIX	Solaris
pthread_sigmask()	thr_sigsetmask()
pthread_kill()	thr_kill()

Functions Related to IDs

POSIX	Solaris
pthread_self()	thr_self()
pthread_equal()	—
—	thr_main()

Functions Related to Scheduling

POSIX	Solaris
—	thr_yield()
—	thr_suspend()
—	thr_continue()
pthread_setconcurrency()	thr_setconcurrency()
pthread_getconcurrency()	thr_getconcurrency()
pthread_setschedparam()	thr_setprio()
pthread_setschedprio()	thr_setprio()
pthread_getschedparam()	thr_getprio()

Functions Related to Cancellation

POSIX	Solaris
pthread_cancel()	—
pthread_setcancelstate()	—
pthread_setcanceltype()	—
pthread_testcancel()	—
pthread_cleanup_pop()	—
pthread_cleanup_push()	—

Functions Related to Mutexes

POSIX	Solaris
pthread_mutex_init()	mutex_init()

pthread_mutexattr_init()	–
pthread_mutexattr_setpshared()	–
pthread_mutexattr_getpshared()	–
pthread_mutexattr_setprotocol()	–
pthread_mutexattr_getprotocol()	–
pthread_mutexattr_setprioceiling()	–
pthread_mutexattr_getprioceiling()	–
pthread_mutexattr_settype()	–
pthread_mutexattr_gettype()	–
pthread_mutexattr_setrobust()	–
pthread_mutexattr_getrobust()	–
pthread_mutexattr_destroy()	–
pthread_mutex_setprioceiling()	–
pthread_mutex_getprioceiling()	–
pthread_mutex_lock()	mutex_lock()
pthread_mutex_trylock()	mutex_trylock()
pthread_mutex_unlock()	mutex_unlock()
pthread_mutex_destroy()	mutex_destroy()

Functions Related to Condition Variables

POSIX	Solaris
pthread_cond_init()	cond_init()
pthread_condattr_init()	–
pthread_condattr_setpshared()	–
pthread_condattr_getpshared()	–
pthread_condattr_destroy()	–
pthread_cond_wait()	cond_wait()
pthread_cond_timedwait()	cond_timedwait()
pthread_cond_signal()	cond_signal()
pthread_cond_broadcast()	cond_broadcast()

pthread_cond_destroy()	cond_destroy()

Functions Related to Reader/Writer Locking

POSIX	Solaris
pthread_rwlock_init()	rwlock_init()
pthread_rwlock_rdlock()	rw_rdlock()
pthread_rwlock_tryrdlock()	rw_tryrdlock()
pthread_rwlock_wrlock()	rw_wrlock()
pthread_rwlock_trywrlock()	rw_trywrlock()
pthread_rwlock_unlock()	rw_unlock()
pthread_rwlock_destroy()	rwlock_destroy()
pthread_rwlockattr_init()	—
pthread_rwlockattr_destroy()	—
pthread_rwlockattr_getpshared()	—
pthread_rwlockattr_setpshared()	—

Functions Related to Semaphores

POSIX	Solaris
sem_init()	sema_init()
sem_open()	—
sem_close()	—
sem_wait()	sema_wait()
sem_trywait()	sema_trywait()
sem_post()	sema_post()
sem_getvalue()	—
sem_unlink()	—
sem_destroy()	sema_destroy()

Functions Related to fork( ) Clean Up

POSIX	Solaris
pthread_atfork()	—

Functions Related to Limits

POSIX	Solaris
pthread_once()	–

Functions Related to Debugging

POSIX	Solaris
–	thr_stksegment()

Locking

Synchronization

Multithreaded behavior is asynchronous, and therefore, optimized for concurrent and parallel processing. As threads, always from within the same process and sometimes from multiple processes, share global data with each other, they are not guaranteed exclusive access to the shared data at any point in time. Securing mutually exclusive access to shared data requires synchronization among the threads. Both POSIX and Solaris implement four synchronization mechanisms: mutexes, condition variables, reader/writer locking (*optimized frequent-read occasional-write mutex*), and semaphores.

Synchronizing multiple threads diminishes their concurrency. The coarser the grain of synchronization, that is, the larger the block of code that is locked, the lesser the concurrency.

MT `fork()`

If a threads program calls `fork(2)`, it implicitly calls `fork1(2)`, which replicates only the calling thread. Should there be any outstanding mutexes throughout the process, the application should call `pthread_atfork(3C)` to wait for and acquire those mutexes prior to calling `fork()`.

Scheduling

POSIX Threads

Solaris supports the following three POSIX scheduling policies:

`SCHED_OTHER` Traditional Timesharing scheduling policy. It is based on the timesharing (TS) scheduling class.

`SCHED_FIFO` First-In-First-Out scheduling policy. Threads scheduled to this policy, if not preempted by a higher priority, will proceed until completion. Such threads are in real-time (RT) scheduling class. The calling process must have a effective user ID of `0`.

`SCHED_RR` Round-Robin scheduling policy. Threads scheduled to this policy, if not preempted by a higher priority, will execute for a time period determined by the system. Such threads are in real-time (RT) scheduling class and the calling process must have a effective user ID of `0`.

In addition to the POSIX-specified scheduling policies above, Solaris also supports these scheduling policies:

`SCHED_IA` Threads are scheduled according to the Inter-Active Class (IA) policy as described in `priocntl(2)`.

`SCHED_FSS` Threads are scheduled according to the Fair-Share Class (FSS) policy as described in `priocntl(2)`.

`SCHED_FX` Threads are scheduled according to the Fixed-Priority Class (FX) policy as described in `priocntl(2)`.

Solaris Threads Only scheduling policy supported is `SCHED_OTHER`, which is timesharing, based on the TS scheduling class.

Errors In a multithreaded application, `EINTR` can be returned from blocking system calls when another thread calls `forkall(2)`.

Usage

`-mt` compiler option The `-mt` compiler option compiles and links for multithreaded code. It compiles source files with `–D_REENTRANT` and augments the set of support libraries properly.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe, Fork 1-Safe

See Also `crle(1)`, `fork(2)`, `priocntl(2)`, `libpthread(3LIB)`, `librt(3LIB)`, `libthread(3LIB)`, `pthread_atfork(3C)`, `pthread_create(3C)`, `attributes(5)`, `standards(5)`

Linker and Libraries Guide

Name trusted_extensions – Solaris Trusted Extensions

Description Solaris™ Trusted Extensions software is a specific configuration of the Solaris Operating System (Solaris OS). Solaris Trusted Extensions (Trusted Extensions) provides labels for local objects and processes, for the desktop and windowing system, for zones and file systems, and for network communications. These labels are used to implement a Multilevel Security (MLS) policy that restricts the flow of information based on label relationships. In contrast to Discretionary Access Control (DAC) based on ownership, the MLS policy enforced by Trusted Extensions is an example of Mandatory Access Control (MAC).

By default, Trusted Extensions software is disabled. It is enabled and disabled (but not configured) by the `labeld(1M)` service, identified by the FMRI:

```
svc:/system/labeld:default
```

Refer to the Administrator's Guide listed below for the required configuration of Trusted Extensions software necessary before use. The system must be rebooted after enabling or disabling `labeld` to activate or deactivate Trusted Extensions software.

See Also `labeld(1M)`, `label_encodings(4)`, `labels(5)`

Solaris Trusted Extensions Administrator's Procedures

Solaris Trusted Extensions User's Guide

Name vgrindefs – vgrind's language definition data base

Synopsis `/usr/lib/vgrindefs`

Description `vgrindefs` contains all language definitions for `vgrind(1)`. Capabilities in `vgrindefs` are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list. Entries may continue onto multiple lines by giving a \ as the last character of a line. Lines starting with # are comments.

Capabilities The following table names and describes each capability.

Name	Type	Description
ab	str	Regular expression for the start of an alternate form comment
ae	str	Regular expression for the end of an alternate form comment
bb	str	Regular expression for the start of a block
be	str	Regular expression for the end of a lexical block
cb	str	Regular expression for the start of a comment
ce	str	Regular expression for the end of a comment
id	str	String giving characters other than letters and digits that may legally occur in identifiers (default '_')
kw	str	A list of keywords separated by spaces
lb	str	Regular expression for the start of a character constant
le	str	Regular expression for the end of a character constant
oc	bool	Present means upper and lower case are equivalent
pb	str	Regular expression for start of a procedure
pl	bool	Procedure definitions are constrained to the lexical level matched by the 'px' capability
px	str	A match for this regular expression indicates that procedure definitions may occur at the next lexical level. Useful for lisp-like languages in which procedure definitions occur as subexpressions of defuns.
sb	str	Regular expression for the start of a string
se	str	Regular expression for the end of a string
tc	str	Use the named entry as a continuation of this one
tl	bool	Present means procedures are only defined at the top lexical level

Regular Expressions

`vgrindefs` uses regular expressions similar to those of `ex(1)` and `lex(1)`. The characters '^', '$', ':', and '\' are reserved characters and must be 'quoted' with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

`$`	The end of a line
`^`	The beginning of a line
`\d`	A delimiter (space, tab, newline, start of line)
`\a`	Matches any string of symbols (like '.*' in lex)
`\p`	Matches any identifier. In a procedure definition (the 'pb' capability) the string that matches this symbol is used as the procedure name.
`()`	Grouping
`|`	Alternation
`?`	Last item is optional
`\e`	Preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) that can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like '(tramp|steamer)flies?' would match 'tramp', 'steamer', 'trampflies', or 'steamerflies'. Contrary to some forms of regular expressions, `vgrindef` alternation binds very tightly. Grouping parentheses are likely to be necessary in expressions involving alternation.

Keyword List

The keyword list is just a list of keywords in the language separated by spaces. If the 'oc' boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

Examples

EXAMPLE 1 A sample program.

The following entry, which describes the C language, is typical of a language entry.

```
C|c|the C programming language:\
    :pb=^\d?*?\d?\p\d?(\a?\)(\d|{):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
    :le=\e':tl:\
    :kw=asm auto break case char continue default do double else enum\
    extern float for fortran goto if int long register return short\
    sizeof static struct switch typedef union unsigned void while #define\
    #else #endif #if #ifdef #ifndef #include #undef # define endif\
    ifdef ifndef include undef defined:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to `vgrind(1)` as 'c' or 'C'.

Files `/usr/lib/vgrindefs` file containing vgrind descriptions

See Also ex(1), lex(1), troff(1), vgrind(1)

Name wbem – Web-Based Enterprise Management

Description Web-Based Enterprise Management (WBEM) is a set of management and Internet-related technologies intended to unify the management of enterprise computing environments. Developed by the Distributed Management Task Force (DMTF), WBEM enables organizations to deliver an integrated set of standards-based management tools that support and promote World Wide Web technology. The DMTF has developed a set of standards that make up WBEM. This set of standards includes:

Common Information Model (CIM) CIM is an object-oriented data model that describes the overall management of information in an enterprise network environment. CIM consists of a CIM specification and a CIM schema:

CIM Specification — Consists of the language and methodology that describes management data.

CIM Schema — Provides actual model descriptions of systems, applications, large area networks, and devices. The CIM Schema enables applications from different developers on different platforms to describe management data in a standard format. As a result, a variety of management applications can share this information.

CIM Operations Over HyperText Transport Protocol (HTTP) 1.1 is a transport mechanism that maps CIM operations to HTTP to allow implementations of CIM to interoperate in an open, standardized manner.

CIM Operations Over HTTP 1.1 uses eXtensible Markup Language (XML), which is a markup language that represents management information in textual form.

In addition to the XML representation, CIM information is also represented textually by the managed object format (MOF). These MOF representations are typically stored as text files that developers compile into a CIM Object Manager.

WBEM Tools and Services Tools and services that enable developers to create and Services management applications and instrumentation that manage heterogeneous computer environments include:

- Solaris WBEM Services
- Solaris WBEM Software Development Kit (SDK)

Solaris WBEM Services These services consist of a set of value-added Services components. These services make it easier for developers to create management applications that run in the Solaris operating environment. They also make the Solaris operating environment easier to manage. Solaris WBEM Services consists of:

- CIM Object Manager, CIM Repository, and MOF Compiler

- CIM and Solaris Schema, which is an extension schema of CIM. CIM and Solaris Schema is a collection of CIM classes that describe managed elements in the Solaris operating environment. These classes are available from the CIM Object Manager at start up.
- Solaris Providers, which are programs that communicate information between the Solaris operating environment and the CIM Object Manager (providers get and set dynamic information about managed elements, acting as an intermediary between the CIM Object Manager and the managed elements).

  Solaris software providers have been developed for a variety of areas: users, roles, file systems, and network configuration, for example. A remote provider is also available to distribute agents away from the CIM Object Manager when required. Because of the incremental development capabilities of the WBEM instrumentation framework, developers can progressively and consistently add more providers for additional Solaris software services.
- SNMP Adapter for WBEM, which enables Simple Network Management Protocol (SNMP) management applications to access system management information that is provided by Solaris WBEM Services. Used with the Solstice Enterprise Agent (SEA) Master Agent `snmpdx(1M)`, the SNMP Adapter for WBEM maps SNMP requests into equivalent WBEM Common Information Model (CIM) properties or instances.

  The SNMP Adapter for WBEM also remaps the response from the CIM Object Manager into an SNMP response, which is returned to the management application.

  A mapping file contains the corresponding Object Identifier (OID), class name, property name, and Abstract Syntax Notation One (ASN.1) type for each object. Developers can create their own mapping files.
- SNMP Provider, which enables WBEM services to deliver SNMP information.

Solaris WBEM SDK

The Solaris WBEM SDK is a set of application programming interfaces (APIs) that contain the components necessary to write management applications. These applications communicate with WBEM-enabled management devices by using XML and HTTP communication standards.

Solaris WBEM applications request information or services from the Common Information Model (CIM) Object Manager through the WBEM APIs. These APIs represent CIM objects as Java classes. The APIs are used to describe managed objects and to retrieve information about managed objects in a system environment. The advantage of modeling managed resources by using CIM is that those objects can be shared across any system that is CIM-compliant.

For more information on the Solaris WBEM SDK, see the *Solaris WBEM Developer's Guide*. The Solaris WBEM API documentation is available in Javadoc format with the Solaris OS installation at `/usr/sadm/lib/wbem/doc/index.html`.

Compatibility of Solaris WBEM Services with Existing Protocols

Adapters and converters enable Solaris WBEM Services of Solaris to work compatibly with existing protocols by mapping WBEM information to these protocols. One such protocol is Simple Network Management Protocol (SNMP).

Legacy management applications can administer WBEM-enabled software in the Solaris operating environment. Developers can write agents or providers that convert information from these protocols to WBEM, and they can write adapters that convert WBEM information into these protocols.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC and x86
Architecture	SUNWwbapi, SUNWwbcor, SUNWwbcou, SUNWwbdev, SUNWwbdoc, SUNWwbpro
CSI	Enabled

See Also

`appletviewer(1)`, `cimworkshop(1M)`, `init.wbem(1M)`, `mofcomp(1M)`, `mofreg(1M)`, `snmpdx(1M)`, `wbemadmin(1M)`, `wbemconfig(1M)`, `wbemlogviewer(1M)`, `attributes(5)`

Name xVM, xvm – Solaris x86 virtual machine monitor

Description The Solaris xVM software (hereafter xVM) is a virtualization system, or *virtual machine monitor*, for the Solaris operating system on x86 systems. Solaris xVM enables you to run multiple virtual machines simultaneously on a single physical machine, often referred to as the *host*. Each virtual machine, known as a *domain*, runs its own complete and distinct operating system instance, which includes its own I/O devices. This differs from zone-based virtualization, in which all zones shares the same kernel. (See `zones(5)`.)

Each domain has a name and a UUID. Domains can be renamed but typically retain the same UUID. A domain ID is an integer that is specific to a running instance. It changes on every boot of the guest domain. Non-running domains do not have a domain ID.

The xVM hypervisor is responsible for controlling and executing each of the domains and runs with full privileges. xVM also includes control tools for management of the domains. These tools run under a specialized control domain, often referred to as *domain 0*.

To run Solaris xVM, use the Hypervisor Visual Panel or the xVM milestone service, `svc:/milestone/xvm`, to configure and enable the hypervisor. This configures the `grub(5)` menu so that, the next time the system is booted, the hypervisor will be active. It is still possible to configure the `grub(5)` menu manually, if required, although manually edited entries will be modified if the Hypervisor Panel or xVM milestone is used. Rebooting will place the system under the control of the hypervisor, which in turn boots the control domain. The control domain is a version of Solaris modified to run under the xVM hypervisor. At the point at which the control domain is running, the control tools are enabled. In most other respects, the domain 0 instance behaves just like a "normal" instance of the Solaris operating system.

The xVM hypervisor delegates control of the physical devices present on the machine to domain 0. Thus, by default, only domain 0 has access to such devices. The other *guest* domains running on the host are presented with virtualized devices with which they interact as they would physical devices.

The xVM hypervisor schedules running domains (including domain 0) onto the set of physical CPUs as configured. The xVM scheduler is constrained by domain configuration (the number of virtual CPUs allocated, and so forth) as well as any run-time configuration, such as pinning virtual CPUs to physical CPUs.

The default domain scheduler in xVM is the credit scheduler. This is a work-conserving, fair-share domain scheduler; it balances virtual CPUs of domains across the allowed set of physical CPUs according to workload.

xVM supports two modes of virtualization. In the first, the operating system is aware that it is running under xVM. This mode is called *paravirtualization*. In the second mode, called *fully virtualized*, the guest operating system is not aware that it is running under xVM. A fully virtualized guest domain is sometimes referred to as a *Hardware-assisted Virtual Machine* (HVM). A variation on a Hardware-assisted Virtual Machine is to use paravirtualized drivers for improved performance. This variation is abbreviated as HVM + PV.

With paravirtualization, each device, such as a networking interface, is presented as a fully virtual interface, and specific drivers are required for it. Each virtual device is associated with a physical device and the driver is split into two. A *frontend* driver runs in the guest domain and communicates over a virtual data interface to a *backend* driver. The *backend* driver currently runs in domain 0 and communicates with both the frontend driver and the physical device underneath it. Thus, a guest domain can make use of a network card on the host, store data on a host disk drive, and so forth.

Solaris xVM currently supports two main split drivers used for I/O. Networking is done by means of the `xnb` (xVM Networking Backend) drivers. Guest domains, whether Solaris or another operating system, use `xnb` to transmit and receive networking traffic. Typically, a physical NIC is used for communicating with the guest domains, either shared or dedicated. Solaris xVM provides networking access to guest domains by means of MAC-based virtual network switching.

Block I/O is provided by the `xdb` (xVM Disk Backend) driver, which provides virtual disk access to guest domains. In the control domain, the disk storage can be in a file, a ZFS volume, or a physical device.

Using ZFS volumes as the virtual disk for your guest domains allows you to take a snapshot of the storage. As such, you can keep known-good snapshots of the guest domain OS installation, and revert the snapshot (using `zfs rollback`) if the domain has a problem. The `zfs clone` command can be used for quick provisioning of new domains. For example, you might install Solaris as a guest domain, run `sys-unconfig(1M)`, then clone that disk image for use in new Solaris domains. Installing Solaris in this way requires only a configuration step, rather than a full install.

When running as a guest domain, Solaris xVM uses the `xnf` and `xdf` drivers to talk to the relevant backend drivers. In addition to these drivers, the Solaris console is virtualized when running as a guest domain; this driver interacts with the `xenconsoled(1M)` daemon running in domain 0 to provide console access.

A given system can have both paravirtualized and fully virtualized domains running simultaneously. The control domain must always run in paravirtualized mode, because it must work closely with the hypervisor layer.

As guest domains do not share a kernel, xVM does not require that every guest domain be Solaris. For paravirtualized mode and for all types of operating systems, the only requirement is that the operating system be modified to support the virtual device interfaces.

Fully-virtualized guest domains are supported under xVM with the assistance of virtualization extensions available on some x86 CPUs. These extensions must be present and enabled; some BIOS versions disable the extensions by default.

In paravirtualized mode, Solaris identifies the platform as `i86xpv`, as seen in the return of the following `uname(1)` command:

```
% uname -i
i86xpv
```

Generally, applications do not need to be aware of the platform type. It is recommended that any ISA identification required by an application use the `-p` option (for ISA or processor type) to `uname`, rather than the `-i` option, as shown above. On x86 platforms, regardless of whether Solaris is running under xVM, `uname -p` always returns `i386`.

You can examine the `domcaps` driver to determine whether a Solaris instance is running as domain 0:

```
# cat /dev/xen/domcaps
control_d
```

Note that the `domcaps` file might also contain additional information. However, the first token is always `control_d` when running as domain 0.

xVM hosts provide a service management facility (SMF) service (see `smf(5)`) with the FMRI:

```
svc:/system/xvm/domains:default
```

...to control auto-shutdown and auto-start of domains. By default, all running domains are shutdown when the host is brought down by means of `init 6` or a similar command. This shutdown is analogous to entering:

```
# virsh shutdown mydomain
```

A domain can have the setting:

```
on_xend_stop=ignore
```

...in which case the domain is not shut down even if the host is. Such a setting is the effective equivalent of:

```
# virsh destroy mydomain
```

If a domain has the setting:

```
on_xend_start=start
```

...then the domain is automatically booted when the xVM host boots. Disabling the SMF service by means of `svcadm(1M)` disables this behavior for all domains.

Solaris xVM is partly based on the work of the open source Xen community.

Control Tools The control tools are the utilities shipped with Solaris xVM that enable you to manage xVM domains. These tools interact with the daemons that support xVM: `xend(1M)`, `xenconsoled(1M)`, and `xenstored(1M)`, each described in its own man page. The daemons are, in turn, managed by the SMF.

You install new guest domains with the command line `virt-install(1M)` or the graphical interface, `virt-manager`.

The main interface for command and control of both xVM and guest domains is `virsh(1M)`. Users should use `virsh(1M)` wherever possible, as it provides a generic and stable interface for controlling virtualized operating systems. However, some xVM operations are not yet implemented by `virsh`. In those cases, the legacy utility `xm(1M)` can be used for detailed control.

The configuration of each domain is stored by `xend(1M)` after it has been created by means of `virt-install`, and can be viewed using the `virsh dumpxml` or `xm list -l` commands. Direct modification of this configuration data is not recommended; the command-line utility interfaces should be used instead.

Solaris xVM supports live migration of domains by means of the `xm migrate` command. This allows a guest domain to keep running while the host running the domain transfers ownership to another physical host over the network. The remote host must also be running xVM or a compatible version of Xen, and must accept migration connections from the current host (see `xend(1M)`).

For migration to work, both hosts must be able to access the storage used by the `domU` (for example, over iSCSI or NFS), and have enough resources available to run the migrated domain. Both hosts must currently reside on the same subnet for the guest domain networking to continue working properly. Also, both hosts should have similar hardware. For example, migrating from a host with AMD processors to one with Intel processors can cause problems.

Note that the communication channel for migration is not a secure connection.

Glossary

backend
: Describes the other half of a virtual driver from the *frontend* driver. The backend driver provides an interface between the virtual device and an underlying physical device.

control domain
: The special guest domain running the control tools and given direct hardware access by the hypervisor. Often referred to as *domain 0*.

domain 0
: See *control domain*.

frontend
: Describes a virtual device and its associated driver in a guest domain. A frontend driver communicates with a *backend* driver hosted in a different guest domain.

full virtualization
: Running an unmodified operating system with hardware assistance.

guest domain
: A virtual machine instance running on a host. Unless the guest is domain 0, such a domain is also called a *domain-U*, where *U* stands for “unprivileged”.

host
The physical machine running xVM.

Hardware-assisted Virtual Machine (HVM)
A fully-virtualized guest domain.

node
The name used by the `virsh`(1M) utility for a host.

virtual machine monitor (VMM)
Hypervisor software, such as xVM, that manages multiple domains.

See Also `uname`(1), `dladm`(1M), `svcadm`(1M), `sys-unconfig`(1M), `virsh`(1M), `virt-install`(1M), `xend`(1M), `xenconsoled`(1M), `xenstored`(1M), `xm`(1M), `zfs`(1M), `attributes`(5), `grub`(5), `live_upgrade`(5), `smf`(5), `zones`(5)

Notes Any physical Ethernet datalink (as shown by `dladm show-phys`) can be used to network guest domains.

Name zones – Solaris application containers

Description The zones facility in Solaris provides an isolated environment for running applications. Processes running in a zone are prevented from monitoring or interfering with other activity in the system. Access to other processes, network interfaces, file systems, devices, and inter-process communication facilities are restricted to prevent interaction between processes in different zones.

The privileges available within a zone are restricted to prevent operations with system-wide impact. See `privileges(5)`.

You can configure and administer zones with the `zoneadm(1M)` and `zonecfg(1M)` utilities. You can specify the configuration details a zone, install file system contents including software packages into the zone, and manage the runtime state of the zone. You can use the `zlogin(1)` to run commands within an active zone. You can do this without logging in through a network-based login server such as `in.rlogind(1M)` or `sshd(1M)`.

The autobooting of zones is enabled and disabled by the zones service, identified by the FMRI:

svc:/system/zones:default

See `zoneadm(1M)`. Note that a zone has an `autoboot` property, which can be set to `true` (always autoboot). However, if the zones service is disabled, autoboot will not occur, regardless of the setting of the autoboot property for a given zone. See `zonecfg(1M)`.

An alphanumeric name and numeric ID identify each active zone. Alphanumeric names are configured using the `zonecfg(1M)` utility. Numeric IDs are automatically assigned when the zone is booted. The `zonename(1)` utility reports the current zone name, and the `zoneadm(1M)` utility can be used to report the names and IDs of configured zones.

A zone can be in one of several states:

`CONFIGURED`	Indicates that the configuration for the zone has been completely specified and committed to stable storage.
`INCOMPLETE`	Indicates that the zone is in the midst of being installed or uninstalled, or was interrupted in the midst of such a transition.
`INSTALLED`	Indicates that the zone's configuration has been instantiated on the system: packages have been installed under the zone's root path.
`READY`	Indicates that the “virtual platform” for the zone has been established. For instance, file systems have been mounted, devices have been configured, but no processes associated with the zone have been started.
`RUNNING`	Indicates that user processes associated with the zone application environment are running.

SHUTTING_DOWN
DOWN — Indicates that the zone is being halted. The zone can become stuck in one of these states if it is unable to tear down the application environment state (such as mounted file systems) or if some portion of the virtual platform cannot be destroyed. Such cases require operator intervention.

Process Access Restrictions

Processes running inside a zone (aside from the global zone) have restricted access to other processes. Only processes in the same zone are visible through `/proc` (see `proc(4)` or through system call interfaces that take process IDs such as `kill(2)` and `priocntl(2)`. Attempts to access processes that exist in other zones (including the global zone) fail with the same error code that would be issued if the specified process did not exist.

Privilege Restrictions

Processes running within a non-global zone are restricted to a subset of privileges, in order to prevent one zone from being able to perform operations that might affect other zones. The set of privileges limits the capabilities of privileged users (such as the super-user or root user) within the zone. The list of privileges available within a zone can be displayed using the `ppriv(1)` utility. For more information about privileges, see `privileges(5)`.

Device Restrictions

The set of devices available within a zone is restricted, to prevent a process in one zone from interfering with processes in other zones. For example, a process in a zone should not be able to modify kernel memory using `/dev/kmem`, or modify the contents of the root disk. Thus, by default, only a few pseudo devices considered safe for use within a zone are available. Additional devices can be made available within specific zones using the `zonecfg(1M)` utility.

The device and privilege restrictions have a number of effects on the utilities that can run in a non-global zone. For example, the `eeprom(1M)`, `prtdiag(1M)`, and `prtconf(1M)` utilities do not work in a zone since they rely on devices that are not normally available.

Brands

A zone may be assigned a brand when it is initially created. A branded zone is one whose software does not match that software found in the global zone. The software may include Solaris software configured or laid out differently, or it may include non-Solaris software. The particular collection of software is called a “brand” (see `brands(5)`). Once installed, a zone's brand may not be changed unless the zone is first uninstalled.

File Systems

Each zone has its own section of the file system hierarchy, rooted at a directory known as the zone root. Processes inside the zone can access only files within that part of the hierarchy, that is, files that are located beneath the zone root. This prevents processes in one zone from corrupting or examining file system data associated with another zone. The `chroot(1M)` utility can be used within a zone, but can only restrict the process to a root path accessible within the zone.

In order to preserve file system space, sections of the file system can be mounted into one or more zones using the read-only option of the `lofs(7FS)` file system. This allows the same file system data to be shared in multiple zones, while preserving the security guarantees supplied by zones.

NFS and autofs mounts established within a zone are local to that zone; they cannot be accessed from other zones, including the global zone. The mounts are removed when the zone is halted or rebooted.

Networking

A zone has its own port number space for TCP, UDP, and SCTP applications and typically one or more separate IP addresses (but some configurations of Trusted Extensions share IP address(es) between zones).

For the IP layer (IP routing, ARP, IPsec, IP Filter, and so on) a zone can either share the configuration and state with the global zone (a shared-IP zone), or have its distinct IP layer configuration and state (an exclusive-IP zone).

If a zone is to be connected to the same datalink, that is, be on the same IP subnet or subnets as the global zone, then it is appropriate for the zone to use the shared IP instance.

If a zone needs to be isolated at the IP layer on the network, for instance being connected to different VLANs or different LANs than the global zone and other non-global zones, then for isolation reasons the zone should have its exclusive IP.

A shared-IP zone is prevented from doing certain things towards the network (such as changing its IP address or sending spoofed IP or Ethernet packets), but an exclusive-IP zone has more or less the same capabilities towards the network as a separate host that is connected to the same network interface. In particular, the superuser in such a zone can change its IP address and spoof ARP packets.

The shared-IP zones are assigned one or more network interface names and IP addresses in `zonecfg(1M)`. The network interface name(s) must also be configured in the global zone.

The exclusive-IP zones are assigned one or more network interface names in `zonecfg(1M)`. The network interface names must be exclusively assigned to that zone, that is, it (or they) can not be assigned to some other running zone, nor can they be used by the global zone.

The full IP-level functionality in the form of DHCP client, IPsec and IP Filter, is available in exclusive-IP zones and not in shared-IP zones.

Host Identifiers

A zone is capable of emulating a 32-bit host identifier, which can be configured via `zonecfg(1M)`, for the purpose of system consolidation. If a zone emulates a host identifier, then commands such as `hostid(1)` and `sysdef(1M)` as well as C interfaces such as `sysinfo(2)` and `gethostid(3C)` that are executed within the context of the zone will display or return the zone's emulated host identifier rather than the host machine's identifier.

Attributes

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu

See Also hostid(1), zlogin(1), zonename(1), in.rlogind(1M), sshd(1M), sysdef(1M), zoneadm(1M), zonecfg(1M), kill(2), priocntl(2), sysinfo(2), gethostid(3C), getzoneid(3C), ucred_get(3C), proc(4), attributes(5), brands(5), privileges(5), crgetzoneid(9F)